Model Tree Forests

by

Phenyo Phemelo Moletsane

Submitted in partial fulfillment of the requirements for the degree
Master of Information Technology (Big Data Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

February 2019

Publication data:

Phenyo Phemelo Moletsane. Model Tree Forests. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, February 2019.

Electronic, hyperlinked versions of this dissertation are available online, as Adobe PDF files, at:

http://cirg.cs.up.ac.za/

http://upetd.up.ac.za/UPeTD.htm

Model Tree Forests

by

Phenyo Phemelo Moletsane
E-mail: moletsanephenyo232@gmail.com

Abstract

A model tree is a decision tree, where the leaf nodes are a model of the independent features. Like regression trees and classification trees, model trees are induced by greedy algorithms which often lead to sub-optimal trees and overfitting of the training data. In an attempt to overcome the problems caused by decision tree induction algorithms, many works have proposed random forests. A random forest is a powerful ensemble method which grows a collection of decision trees whose predictions are aggregated. While ensembles of classification trees have been widely studied and proven to improve the predictive performance of classification trees and to avoid overfitting, ensembles of model trees have received little attention. This work investigates a model tree ensemble approach, referred to as model tree forests. In particular, the M5' model tree algorithm was used as a base learner for the model tree forest. Firstly, the performance of the M5' model tree forests was compared to the performance of the single-induced M5' model tree. Secondly, the M5' model tree forest was analyzed with reference to the number of trees and the number of variables randomly selected for splitting at each node of the tree. Experimental results showed that the M5' model tree forest performed better than the single induced M5' model tree.

Keywords: Model tree forests, model trees, decision trees, random forests.

Supervisors : Prof. A. P. Engelbrecht
Dr. J. Grobler
Department : Department of Computer Science
Degree : Master of Information Technology

Acknowledgements

I would like to express my sincere gratitude to the following people for their assistance during the completion of this dissertation:

- Prof. Andries P. Engelbrecht for his valuable guidance, support and constant encouragement through the process of completing this work.
- Dr. Jacomine Grobler for her heartfelt support and assistance in keeping my progress on schedule and proofreading my work.
- The University of Pretoria for giving me the opportunity to pursue this research work.
- My family for their love, support and encouragement.
- My late father for always believing in me.
- My Creator for giving me the strength to complete this research.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

Decision trees are one of the most popular predictive models which can be applied to both regression and classification problems. Recent years have seen a growth in their popularity owing to their simplicity and computational efficiency [62], [67]. The variants of decision trees include classification trees, regression trees, and model trees.

A model tree can be described as a variant of decision trees where the leaf nodes are a model of the independent features of the problem, used to make predictions for the patterns that match the conditions followed from the root node to the leaf node of the tree [60]. Model trees have been widely used in many applications and proved to be successful in predicting a target variable that takes on a continuous numeric value [60], [74]. Model trees are very similar to regression trees, but instead of having a single discrete value in the leaf nodes, they have functions, or models, in their leaf nodes [60], [74].

A typical model tree consists of internal nodes that denote tests on attributes, and branches that represent the outcome of the tests. The leaves hold models, which are either linear or nonlinear, at the bottom of the tree [33]. The prediction of the target value is obtained by traversing down the tree until a leaf is reached and then using the corresponding model to estimate the response variable [60], [74].

As with classification tree and regression tree induction algorithms, model tree induction algorithms use recursive partitioning to grow the tree [60]. Induction of a model tree begins from the top, or root node, to search for a local optimal split and then continues

by passing on the subsets to the relevant node [60]. This recursive partitioning continues until all the subsets are homogeneous. The use of such algorithms is known to lead to trees that do not generalize the data well and overfit the training data [62]. Moreover, the model trees may be unstable, meaning that slight changes in the data may lead to large changes in the structure of the tree [62].

Several studies have proposed ensemble methods, such as bagging [11] and random forests [12] to mitigate the problems caused by decision tree induction algorithms [12]. Since their introduction, random forests have been successful in reducing overfitting of classification trees and improving generalization of classification trees [12]. A random forest is a collection of trees where each tree is induced by randomly sampling the original data and splitting of each node by using randomly selected features of the data [12]. The final prediction is obtained by aggregating the predictions of the trees in the forest, thus reducing the variance and improving generalization [12].

While ensembles of classification trees have been widely studied and proven to improve the predictive performance of classification trees, ensembles of model trees have received little attention [4]. This dissertation particularly investigates M5' model tree forests for regression problems in an attempt to improve the performance of the M5' model tree. M5' is a well-known model tree induction algorithm developed by Wang and Witten [74]. The M5' algorithm is based on Quinlan's original M5 algorithm. Its novelty results from the use of simple linear regression models in leaf nodes and low computational demands.

1.1 Problem statement

Model tree induction algorithms, such as the M5 model tree induction algorithm, make use of a top-down greedy approach to induce the tree [50]. Such algorithms are sequential in nature and determine the best split by optimizing the relative entropy of the training subsets at each node [7], [47], [62]. Each partition in the tree is selected locally without considering the possible impacts of the following splits in the tree. Induction of model trees by greedy algorithms has proven to be efficient in many practical problems [7]. However, convergence to a global optimal solution is hardly feasible [7]. Thus, the result

of inducing model trees through these algorithms is often sub-optimal model trees that overfit the training data and consequently have low predictive performance on unseen data [7], [47], [62]. Moreover, the top-down induction algorithms often require pruning after growing the tree to its full depth to avoid overfitting the training data. This circumvents the problem of overfitting; however, it means that the training of decision trees occurs in two stages, inducing the tree through greedy algorithms and pruning. Hence, ensemble methods have been developed to solve the overfitting problem in decision trees.

1.2 Objectives

The random forest algorithm, proposed by Breiman in 2001, is one of the ensemble methods which have been widely used to overcome the overfitting problem in decision trees [12]. Because random forests of classification trees have been successful in avoiding overfitting and improving the predictive performance of classification trees, a forest of model trees was proposed. The main objective of this study was to conduct an empirical analysis of M5' model tree forests as an approach to overcome the overfitting problem in model trees.

The following objectives have been defined:

- To develop a forest of M5' model trees for regression data sets where the target value is a continuous numeric value.
- To compare the performance of the M5' model tree forest with that of a single-induced model tree.
- To analyze the M5' model tree forests with respect to the number of trees in the tree and the number of variables randomly selected for splitting at each node.

1.3 Contributions

The main contributions of this dissertation include the following:

- Proposing random forests of M5' model trees.
- The analysis of model tree random forests.

1.4 Thesis outline

The list below provides the organization of the chapters which contribute to this dissertation:

- **Chapter 2** gives the necessary background relating to decision trees and the role of decision trees in data mining. The chapter also discusses the advantages and the disadvantages of decision trees.
- **Chapter 3** provides an overview of the top-down induction method employed for constructing decision trees. The chapter covers splitting criteria, stopping rules overfitting and pruning.
- **Chapter 4** examines the M5 model induction algorithm and its extension, the M5' algorithm. The chapter further discusses other related model tree induction algorithms.
- **Chapter 5** focuses on popular ensemble methods that have been used to improve decision tree performance. The chapter specifically looks at two ensemble methods, namely bagging and random forests.
- **Chapter 6** provides an empirical analysis of model tree forests. The chapter covers all aspects relating to the experimental design including experimental methodology, the data sets used, and analysis of the results.
- **Chapter 7** summarizes the findings of the dissertation and suggests possible future work.

Chapter 2

Decision trees in data mining

The field of data mining has grown rapidly over the last two decades and has become increasingly important in various fields of society. Data mining can simply be defined as the process of extracting of knowledge and information from large amounts of data [28], [75]. One of the earliest definitions of data mining, provided by Frawley et al., was "the non-trivial extraction of implicit, previously unknown and potentially useful information from data" [32]. Owing to its significance in decision making, data mining has become prevalent in many important domains including health care, education, business, finance, science, security, amongst others. Different data mining techniques have been developed and studied in the effort to extract useful information from large data sets [62].

One of the most powerful and popular data mining techniques is decision trees. Decision trees are simple yet successful predictive modelling techniques which can be applied to both regression and classification problems [62]. Due to their ease of interpretation, robustness and ability to handle both numeric and categorical data, decision trees have earned significant and growing interest from many researchers in the machine learning community [7], [47]. The aim of this chapter is to provide an overview of decision trees. Section 2.1 provides a detailed background on decision trees, while Section 2.2 discusses the advantages of decision trees. Section 2.3 analyzes the limitations of decision trees and finally a summary of the chapter is provided in Section 2.4.

2.1 Decision trees: Basic concepts

Decision trees are non-parametric supervised learning techniques that use a set of continuous, binary or categorical predictor variables to either predict a continuous response or to classify a categorical target response [7], [62]. A decision tree creates a model in the form of a tree structure to predict response variables by learning logical decision rules from the data features [7]. Aho et al. describes decision trees as analogous to a graph [1], [7], [63]. Generally, a decision tree can be viewed as a directed graph $G = (U, E)$ that comprises a limited, nonempty set of nodes U and a set of edges E. Such a graph has the following characteristics [1]:

- The graph is said to be directed if the edges are ordered pairs (u, e) of nodes;
- There are no cycles in the graph, i.e., the graph is acyclic;
- An acyclic graph has one node, called the root, with no incoming edges;
- With the exception of the root node, every node has one incoming edge;
- The path from the root node to each node is unique. The path is a sequence of edges of the form $(u_1, u_2), (u_2, u_3), ..., (u_{n-1}, u_n)$;
- If there is a path from node u to e (ue), then u is said to be a proper ancestor of u and e is said to be a proper descendant of u. A terminal node, or a leaf, is a node that has no proper descendant. All the other nodes, except for the root node, are known as internal or test nodes.

In decision trees, the length of the path from the root node to the deepest leaf node is referred to as the depth of the tree, while the number of interior nodes at each level is referred to as the breadth of the tree [1]. The depth and breadth of a decision tree are important measures of tree complexity. Higher values of the depth and breadth of the tree produce more complex trees [7]. Such trees are known to have low predictive performance and may be improved by employing pruning or a stopping criterion (to be discussed in Chapter 3) [7], [62].

A decision tree is hierarchical in nature and can be thought of as a technique to solve a complex problem by performing a series of tests or asking successive questions [7], [13], [62]. At the root and internal nodes, a decision tree partitions a given data set attribute into subsets based on a test that minimizes an error function [47], [75]. The resulting edges correspond to the outcomes of the test. The next test depends on the previous test. The leaf nodes are assigned either a class label (known as classification trees), a continuous value (regression trees), or a model/ function (model tree). The prediction for the response variable is obtained by traversing along the path from the root down the tree until a leaf is reached [7], [62]. When the leaf is reached, the corresponding information is used to obtain the prediction and thus make a decision [7], [62]. Figure 2.1 shows an example of a decision tree that predicts the prices of Boston houses. Leaf nodes are denoted as rectangles while internal nodes are denoted as ovals. In this particular example, the leaf nodes hold a continuous value.

Decision trees can be easily converted into an equivalent set of rules [7], [59], [62], [72]. Each path of the tree from the root to the leaf node is converted into a rule as follows: the internal nodes and their corresponding edges are transformed into the conditions of the rule and the terminal nodes are transformed into the consequent of the rule [62]. For example, one of the paths in Figure 2.1 can be transformed into the rule: "If the value of L_{stat} is less than or equal to 4.65 and the R_m value is less that or equal to 7.37, then the house will have an average price of 34.14".

Quinlan's work presented a technique to transform a classification tree to a set of production rules and pointed out two advantages of converting decision trees into rules [59]. Firstly, conversion of decision trees to rules improves the comprehensibility of the induced tree to a human user. Lastly, and most importantly, converting decision trees into rules can improve classification performance [59].

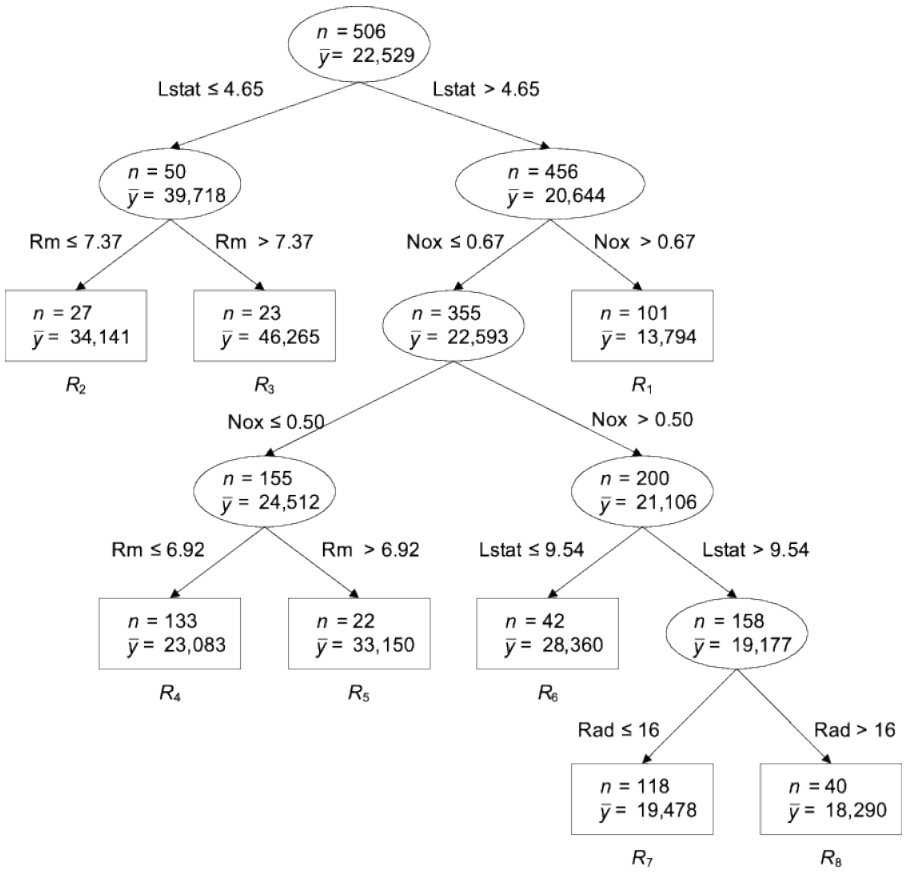


Figure 2.1: Example of a regression tree for Boston housing data.

2.2 Advantages of decision trees

As mentioned earlier, decision trees are among the most popular data mining techniques. Owing to their many attractive advantages such as simplicity and efficiency, decision trees have been successfully applied to a broad range of tasks from diagnosing medical cases to predicting whether or not a potential customer will respond to a banking campaign. The rest of the advantages of decision trees are discussed in detail below [7], [13], [20], [36], [62]:

- Decision trees are simple, easy to understand and easy to use. One of the useful features of decision trees is their ability to break down a complex problem into simpler decisions. Compared to black box models (e.g., artificial neural networks),

decision trees are easier to interpret and can be visualized easily.

- While other models require data sets that have only one type of data variable, tree-based techniques can handle both numerical and categorical data directly. Moreover, decision trees do not require any significant pre-processing of the input data such as imputing missing values, normalization and converting categorical variables to numerical variables. By contrast, other data mining techniques such as linear regression require that the input data be pre-processed before the model is applied to the data.

- Another advantage of decision trees is that they can be used for feature selection. Feature selection is one of the crucial steps in data pre-processing. The aim of feature selection in machine learning tasks is to reduce data dimensionality by eliminating irrelevant attributes so as to improve efficiency [35], [37]. At each interior node of the tree, the feature that results in the lowest estimated error is selected to split the data. Thus, this feature can be regarded as the most important feature [37]. Decision trees are among the many feature selection techniques that have been proposed in the literature. Gayatri et al. studied decision rule induction for feature selection in software prediction defects [37]. The study revealed that using only 15 features out of 94 features produces better classification performance than using the complete set of features [37]. Furthermore, decision tree induction resulted in better performance when compared to relief and support vector machine (SVM) [37].

- Decision trees are non-parametric methods. This means that they do not make any assumptions about the distribution of the data. Moreover, they are highly robust. For example, they readily accommodate outliers, missing values, and data sets that contain errors (i.e. noise).

- Decision trees can be used for multi-target problems. In multi-target problems, the aim is to predict multiple target variables. Several multi-target regression trees have been studied. One of the earliest multi-target regression trees was proposed by De'ath [21]. De'ath's work extended the CART (Cart and Regression Trees) algorithm to a multi-target regression problem and follows the same steps as CART

to build multi-target regression trees. As with univariate trees, multi-target regression trees are easy to build, easy to interpret and can handle missing values. Another multi-target tree algorithm, named Multi-Target Stepwise Model Tree Induction (MTSMOTI), was developed by Appice and Deroski [6]. The MTSMOTI is used to induce multi-target model trees in a stepwise manner. Each leaf node of the MTSMOTI holds several linear functions which predict the values of different target variables [6].

Although decision trees have proven to be powerful and useful, they present several challenges. These challenges are discussed in Section 2.3.

2.3 Limitations of decision trees

The disadvantages of decision trees are listed below:

- Decision trees are prone to overfitting [7], [13], [62]. Overfitting arises when a model learns the data too well that its predictive performance on unseen data is reduced [30]. To overcome overfitting in decision trees, pruning is often applied after the nodes are split [13], [30], [58]. Other solutions to overfitting include building an ensemble of trees, controlling the tree depth, and controlling the minimum number of samples in each leaf node [13].

- While decision trees are generally robust to outliers, they can be very unstable. This means that a slight change in the data set can result in a different tree being generated.

- Constructing an optimal decision tree is known as an 'NP-complete' problem [51]. This means that computation of a globally optimal decision is practically impossible [51]. Consequently, decision tree algorithms are based on greedy algorithms where locally optimal decision trees are achieved at each node of the tree [51]. Such algorithms typically yield locally optimal trees, and do not guarantee globally optimal or best trees.

2.4 Summary

The aim of this chapter was to provide a general background on decision trees. The characteristics and the general structure of decision trees were discussed in Section 2.1. Section 2.2 and Section 2.3 critically analyzed decision trees by highlighting the advantages as well as the disadvantages of decision trees respectively. The next chapter focuses on the method used to construct decision trees as well as popular decision tree induction algorithms.

Chapter 3

Inducing decision trees

Numerous decision tree induction algorithms have been introduced over the years, such as Quinlan's Iterative Dichotomiser (ID3) [58], Breiman et al.'s Classification and Regression Trees (CART) [13]. In general, inducing an optimal decision is considered to be an NP-complete problem. The NP-completeness of decision tree induction has been proven by several papers in the literature. Hyafil and Rivest's work showed that inducing an optimal binary tree, where an optimal tree is one with the minimum average depth, is an NP-complete problem [41]. Murphy and McCraw [51] argue that in most cases, building storage efficient decision trees is an NP-complete problem. Moreover, Hancock et al. proved that constructing a minimal decision tree that is consistent with the training data set, is NP-complete [38]. Consequently, it was necessary to develop heuristics to overcome the problem of inducing decision trees. These heuristics rely on a greedy, top-down strategy to build reasonably accurate, sub-optimal decision trees in a short amount of time [13], [58]. The rest of this chapter discusses the top-down induction algorithm in more detail. The chapter also introduces the most popular splitting criteria for classification trees and regression trees in Section 3.1. The section further discusses the problem of overfitting and the use of pruning in decision trees. Lastly, Section 3.2 examines and analyzes decision tree algorithms.

3.1 Top-down induction algorithm

The goal of this section is to discuss the top-down induction algorithm of decision trees in more detail. The section starts by discussing Hunt's Concept Learning System framework (CLS) as the pioneer work in top-down induction of decision trees. The section further provides an improved up-to-date algorithmic framework for top-down induction of decision trees. The section is divided into sub-sections. The sub-sections focuses on the conceptual phases of the top-down induction algorithm including splitting the nodes, stopping the top-down induction procedure and pruning the tree. Section 3.1.1 discusses the splitting criteria used for inducing decision trees. The most common splitting criteria, such as information theory, distance, dependence and impurity-based criteria, are discussed. Section 3.1.2 focuses on the stopping criteria for decision trees induction and discusses the overfitting phenomenon. Lastly, Section 3.1.3 discusses pruning as well as the pruning methods that are available in the literature.

Most of the existing top-down decision tree induction algorithms are based on Hunt's Concept Learning System [40]. Hunt's algorithm grows a decision tree in a recursive manner by partitioning the training set into successively purer subsets using a cost-driven function [40]. Let S_t be the set of training records at node t and $y = \{y_1, y_2, ..., y_k\}$ be the class labels in a k-class task. The following two steps represent a recursive definition of Hunt's algorithm [40] [63]:

- Step 1: If all the records in S_t belong to the same class y_t, then t is a leaf node labelled as y_t.
- Step 2: If S_t contains records that belong to more than one class, then an attribute test is used to split the records into smaller subsets. For each outcome of the test condition, a child node is created. The records in S_t are passed down to the children based on the outcomes. The algorithm is then recursively applied to each child node.

Many researchers have studied this algorithm; however it has some drawbacks. One of the major problems is that the algorithm is only practical if the training data is consistent, i.e. if each combination of attributes has a unique class. Furthermore, its

stopping criterion, as stated in Step 1 above, is that all terminal nodes must be pure. This stopping criterion can result in decision trees that tend to overfit the training data. Succeeding algorithms such as CART and ID3 aimed to improve Hunt's algorithm. In contrast to Hunt's method, these algorithms use information entropy functions to split the training data into smaller subsets [13], [58]. Algorithm 3.1 demonstrates the improved generic framework for top-down induction of decision trees. The algorithm comprises two steps for inducing a decision tree, namely growing the tree (*TreeGrowing*), and pruning the tree (*TreePruning*) [7], [62].

TreeGrowing(S,A,y,$SplitCriterion$,$StoppingCriterion$) where:
S - Training set
A - Predictive attributes set
y - Target attribute
$SplitCriterion$ - the method for evaluating a split
$StoppingCriterion$ -the criteria used to cease the growing process
Create a new tree T with a single root node.
IF $StoppingCriterion(S)$ **THEN**
    Mark T as a leaf with the most
    common value of y in S as a label.
ELSE
    $\forall a_i \in A$ find a that obtain the best $StoppingCriterion(a_i, S)$
    Label t with with a
    FOR each outcome v_i of a
        Set $Subtree_i = TreeGrowing(\sigma_{a=v_i} S, A, y)$.
        Connect the root node of t_T to $Subtree_i$ with an edge that is labelled as v_i
    END FOR
END IF
RETURN $TreePruning(X, T, y)$

Algorithm 3.1: Generic algorithmic for top-down induction of decision trees.

$TreePruning(S, T, y)$
where:
S - Training set
T - Tree to be pruned
y - Target attribute
DO
    Select a node t in T such that pruning it
    improves some evaluation criteria to a maximum
    IF $t \neq \emptyset$ **THEN** $T = pruned(T, t)$
UNTIL $t = \emptyset$
RETURN T

Algorithm 3.1: Generic algorithmic for top-down induction of decision trees (Cont.).

The training set, S, is described as a bag instance consisting of tuples [62]. A represents a set of input variables comprising n variables, $A = \{a_1, a_2, ..., a_n\}$, and y is the target variable. Variables are either nominal or numeric [62]. When the variable a_i is nominal, its domain values are represented by $dom(a_i) = \{v_{i,1}, v_{i,2}, ..., v_{i,|dom(a_i)|}\}$ where $dom(a_i)$ is the nominal variable's finite cardinality [62]. Similarly, the domain values of the target variable, y is denoted by $dom(y) = \{c_1, c_2, ..., c_{|dom(y)|}\}$ [62]. In contrast, numeric variables have infinite cardinality [62]. For each iteration, the top-down induction algorithm partitions the training data set using the outcome of a function of the input variables. The most appropriate function is selected according to some split criterion (*SplitCriterion*) [62]. After an appropriate split is selected, the training set is further subdivided into smaller subsets until a stopping criteria (*StoppingCriterion*) is met [62].

3.1.1 Splitting criteria

The key step of inducing a decision tree is to choose the best attribute upon which to split a node into subsets so that the resulting nodes are the purest. In most decision trees induction algorithms, splitting criteria are univariate [7], [48], [62]. Univariate splitting criteria are those that split an internal node according to the value of a single attribute

[7], [48], [62]. These criteria can be categorized in different ways such as: information theory, distance, dependence and impurity-based criteria. The following sections describe the most common splitting criteria that exist in the literature.

Impurity-based criterion

The node is said to be pure when all the instances in a node are of the same class. To estimate the purity of a node, a measure called the impurity function is used [7], [62]. Given a classification task with k classes, an impurity function is a non-negative function that is distributed according to $P = (p_1, p_2, ..., p_k)$, where p_k is the probability that a data point belongs to class k in the current sub-tree. Any function, ϕ, must satisfy the following three properties [7], [62]:

- The function $\phi(P)$ has a unique maximum only when the classes in the node are uniformly distributed, that is at point $(1/k, 1/k, ..., 1/k)$.
- The function $\phi(P)$ has a minimum when the node is pure, that is at $\phi(1, 0, ..., 0)$, $\phi(0, 1, 0, ..., 0)$,..., $\phi(0, 0, ..., 0, 1)$.
- The function $\phi(P)$ is symmetric with respect to its arguments $(p_1, p_2, ..., p_k)$.

The impurity function for a node can then be used to evaluate the quality of a given split, also known as the goodness of fit of a split. The goodness of a split can be measured as the reduction in impurity of the target variable [62]. The goodness of fit due to discrete variable a_i after partitioning the instance space A according to the values $v_{i,j} \in dom(a_{ai})$ is defined by the following [62]:

$$\Phi(a_i, S) = \phi(P_y(S)) - \sum_{j=1}^{|dom(a_i)|} \frac{|\sigma_{a_i=v_{i,j}} S|}{|S|} \cdot \phi(P_y(\sigma_{a_i=v_{i,j}} S)) \tag{3.1}$$

where σ denotes a selection of the training set tuples. The following sections provide an overview of the most popular impurity based criteria for classification trees.

Information gain criterion

The information gain measures how much information a variable reveals about a particular class. It makes use of entropy as an impurity measure. The aim of the information gain is to reduce the entropy resulting from splitting each individual node. Information gain is defined by [39], [58], [62]:

$$InformationGain(a_i, S) = Entropy(y, S) - \sum_{v_{i,j} \in dom(a_i)} \frac{|\sigma_{a_i=v_{i,j}} S|}{|S|} (y, \sigma_{a_i=v_{i,j}} S) \quad (3.2)$$

where the entropy is given by:

$$Entropy(y, S) = \sum_{c_j \in dom(y)} \frac{-|\sigma_{y=c_j} S|}{|S|} \cdot log_2 \frac{-|\sigma_{y=c_j} S|}{|S|} \quad (3.3)$$

The disadvantage of information gain is its bias towards multi-valued features [7]. To overcome this challenge, Quinlan proposed the gain ratio which normalizes the information gain by the entropy of the feature being tested [61].

Gain ratio criterion

The gain ratio criterion normalizes the information gain by dividing the information gain by attribute self-entropy, $Entropy(a_i, S)$, as follows:

$$GainRatio(a_i, S) = \frac{InformationGain(a_i, S)}{Entropy(a_i, S)} \quad (3.4)$$

Quinlan showed that the gain ratio performs better than the information gain [58]. Nevertheless, the gain ratio has some drawbacks. The gain ratio may be undefined when the denominator is zero. Moreover, the ratio may favour features with very low self-entropy. To solve these problems, Quinlan suggested that the ratio be calculated in two stages [7], [61]. Firstly, the information gain is estimated for all features. Secondly, the gain ratio is calculated only for cases in which the value of the information gain is above the average value of the information gain of all features [61].

Gini index and Twoing criteria

The Gini index is a distance-based criterion which is also a form of impurity-based criterion. The Gini index measures the distance between the probability distributions of the classes [13], [55]. The Gini index has been used in the popular induction algorithm, CART, and is calculated as [62]:

$$Gini(y, S) = 1 - \sum_{c_j \in dom(y)} \left(\frac{|\sigma_{y=c_j} S|}{|S|} \right)^2 \tag{3.5}$$

The main drawback with the Gini index is that it may be biased towards features with many values [13]. For such cases, Breiman et al. proposed using a binary criterion called the Twoing criterion. The Twoing criterion aims to achieve a more even segregation of data than the Gini index criterion. The key idea of the Twoing criterion is to transfer a multi-class problem into a binary class problem [13]. This criterion is given by [13], [62]:

$$Twoing(a_i, dom_1(a_i), dom_2(a_i), S) = 0.25 \cdot \frac{|\sigma_{a_i \in dom_1(a_i)}\ S|}{|S|} \cdot \frac{|\sigma_{a_i \in dom_2(a_i)}\ S|}{|S|} \cdot$$

$$\left(\sum_{c_i \in dom(y)} \left| \frac{|\sigma_{a_i \in dom_1(a_i)\ AND\ y=c_i}\ S|}{|\sigma_{a_i \in dom_1(a_i)}\ S|} - \frac{|\sigma_{a_i \in dom_2(a_i)\ AND\ y=c_i}\ S|}{|\sigma_{a_i \in dom_2(a_i)}\ S|} \right| \right)^2 \tag{3.6}$$

The major disadvantage of the Twoing criterion it is computationally more complex than the Gini index.

Other splitting criteria for classification trees

Other splitting criteria for classification trees have been proposed in the literature. De Mantras introduced a distance measure (DM) as an alternative to the gain ratio normalization [19]. This criterion is defined by [19], [62]:

$$DM = \frac{\Phi(a_i, S)}{-\sum_{v_{i,j} \in dom(a_i)} \sum_{c_k \in dom(y)} \frac{|\sigma_{a_i = v_{i,j}\ AND\ y=ck}\ S|}{|S|} log_2 \frac{|\sigma_{a_i = v_{i,j}\ AND\ y=ck}\ S|}{|S|}} \tag{3.7}$$

Kearns and Mansour proposed the DKM (Dietterich, Kearns, Mansour) criterion, defined as: [46].

$$DKM(y,S) = 2 \cdot \sqrt{\left(\frac{|\sigma_{y=c_1}\ S|}{|S|}\right) \cdot \left(\frac{|\sigma_{y=c_2}\ S|}{|S|}\right)} \tag{3.8}$$

The DKM criterion is an impurity-based criterion which is used for binary class problems [46]. Compared to the information gain and the Gini index, the DKM criterion has been shown to produce smaller trees to produce a certain error [46], [62].

Many other splitting criteria for classification trees can be found in the literature, including the mean posterior improvement criterion [69], orthogonality criterion [27] and the AUC splitting criterion [29].

Splitting criteria for regression trees

Thus far, the criteria discussed are for classification trees. For regression trees, the most common impurity measure is the variance or sum of squared deviations from the mean of the leaf node. This criterion is also known as the least squares error. Using the variance as the impurity measure is rationalized by the fact that the best feature in a node is the expected value of the predicted feature on the instances that belong to the node. Therefore, the variance is the mean squared error (MSE) of this best feature. Regression trees induced by the CART algorithm use the mean squared error between the observed target variable and its estimated value as an impurity measure [13]. The MSE of a tree T is given by:

$$MSE(T) = \frac{1}{n} \sum_{t \in T} \sum_{x_i \in t} (y_i - \bar{y}_i)^2 \tag{3.9}$$

where n is the number of training instances, (x_i, y_i), and $\bar{y}_i$ is the sample mean of the target variable estimated on the instances in node t. The MSE criterion seeks the split that minimizes the MSE at a node [13]. The criterion of minimizing squared deviations has been used in many existing decision tree induction algorithms including model tree induction algorithms [50]. HTL (Hybrid Tree Learner), RETIS (Regression Tree Induction System) and M5' (to be discussed in Chapter 4) all use minimization of squared errors to partition the feature space. Minimization of squared error may lead

to trees that are affected by the presence of outliers and skewed distributions [13]. It is important to note that the predicted value for the minimization of squared error, is the average of the response feature for the instances associated with a leaf node. Therefore, the presence of outliers can influence the mean, leading to leaf nodes that may not be representative of the corresponding training instances.

To induce trees that are more robust to outliers, Breiman et al. proposed the least absolute deviation (LAD) as a splitting criterion [13]. In contrast to the minimization of squared error criterion, the least absolute deviation criterion holds that the predicted value is the median of the response feature associated with the leaf node [13]. Breiman et al. suggested splitting nodes so that the reduction in the following measure is maximized [13]:

$$LAD(T) = \sum_{t \in T} \sum_{x_i \in t} |y_i - median(y_i)| \tag{3.10}$$

where $median(y_i)$ is the median of the target variable estimated on the instances in node t. The LAD criterion is generally known to be more robust than the minimization of squared deviation criterion. The use of LAD as a splitting criterion for regression trees was shown to generate trees that are more robust to outliers and skewed distributions [16]. Trees that use the LAD criterion have not been widely used because the criterion is considered to be computationally inefficient when compared to trees that use the minimization of squared error criterion [16]. Furthermore, the LAD criterion often leads to large errors compared to minimization of squared differences.

In their work, "Data Mining Criteria for Tree-Based Regression and Classification", Lee and Buja proposed two splitting criteria for regression trees, namely one-sided purity (OSP) and one-sided extremes (OSE) [14]. The aim of these criteria is to find pure nodes with the smallest variance or extreme nodes with the highest mean as soon as possible. Instead of using the sum weighted averages of impurities of the left side and the right side of the split, Lee and Buja's OSP replaces the weighted average impurities with their minimum [14]. This results in finding a split whose left or right side is the single node with the smallest variance or impurity. Thus, this node will not be split again while the nodes with high impurity will be split further [14]. The OSE criterion finds a split whose left or right side is the single node with the highest mean. Lee and Buja's OSE and OSP

are aggressively greedy [7], [14].

Other regression tree splitting criteria in the literature include Alpaydin's worst possible error (WPE) [5] and the mean posterior improvement (MPI) [68].

3.1.2 Stopping criteria and overfitting

The top-down induction procedure continues to grow the tree until a stopping criterion is met. The following criteria are some of the most common criteria in the literature. Tree induction stops when either one of the following rules has been satisfied [26], [62]:

- A specified maximum tree depth has been reached;
- When all instances in a given node are of the same class, or belong to the same response values;
- When the specified minimum number of instances for a non-terminal node is reached;
- When the specified splitting criterion threshold is not exceeded.

All the criteria above can be considered as pre-pruning stages. By stopping the tree growth prematurely, most of these criteria may lead to underfitted trees. At the same time, loose stopping criteria result in very large trees that overfit the training data. To overcome this dilemma, the literature suggests that decision trees be grown to their full depth using a loose criterion [7], [13]. This will then be followed by a post-pruning method to prevent overfitting the training data.

Overfitting is a phenomenon that occurs when the algorithm perfectly learns the training data but fails to generalize well to the unseen data [30], [62]. In other words, instead of learning, the algorithm memorizes the training data including its potential noise. In decision trees, overfitting is usually attributed to many nodes in the tree relative to the size of training data. Figure 3.1 illustrates the overfitting process: The training error and the generalization error are presented as a function of model complexity. As the tree grows bigger, the training error continues to decrease. On the contrary, the generalization error decreases and then starts to increase at a certain number of nodes

due to overfitting. In Figure 3.2 the generalization error is presented as a function of the training data size. As the training data size increases, the generalization error decreases. This is because it is often difficult to generalize when using a small training data size, thus the model over-learns all the training data instances [62].

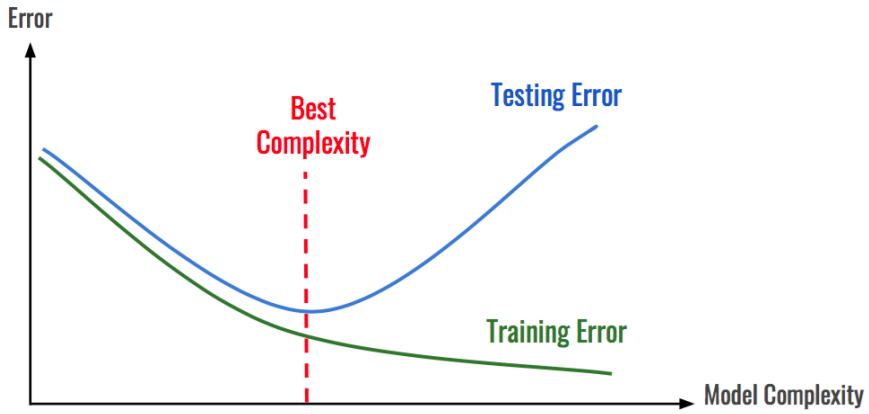


Figure 3.1: Illustration of overfitting in decision trees [62]. The graph shows how the generalization error and the training error change with the number of nodes.

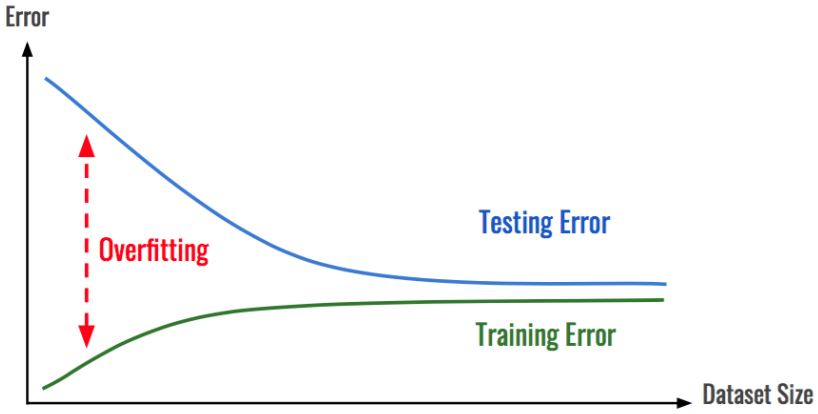


Figure 3.2: Overfitting in decision trees [62]. The graph shows how the error changes with the training set size.

As mentioned earlier, pruning is employed to overcome overfitting. There are various methods for pruning decision trees. The following sub-sections present the most popular pruning techniques for decision trees.

3.1.3 Pruning

Pruning aims to reduce the size of the tree while maintaining or even improving the performance [13]. A tree is fully grown to the extent that it overfits the training data. Then the overfitted tree is reduced to a smaller tree by removing one or more subtrees that are not contributing to the predictive performance [13]. The following sections discuss reduced error pruning, pessimistic error pruning, error-based pruning, and cost-complexity pruning.

Reduced error pruning

Reduced error pruning, proposed by Quinlan [59], is a very simple method for pruning. While traversing over the internal nodes of the tree from the bottom to the top, reduced error pruning evaluates each internal node with regards to the classification error [59], [62]. The procedure checks whether replacing the internal node with the leaf node reduces the generalization error of the tree. If such an error decreases when the internal node is replaced with a leaf node, then the internal node must be pruned [59], [62]. The procedure continues until any further pruning would increase the generalization error. Quinlan proposes using a pruning data set (a partition of the training set) to evaluate each internal node with regards to the error [59], [62]. It has been shown that this pruning method results in the smallest, most accurate tree [59], [62]. The need for the pruning data set, however, divides the training set, resulting in fewer instances (which is a problem for small data set) of training data available to grow the tree.

Pessimistic error pruning

Pessimistic error pruning was also proposed by Quinlan [59], [61]. This technique avoids the need for a pruning data set and thus uses the training data set for both growing and pruning stages. According to Quinlan, the error rate estimated over the training data set is not reliable enough and can not be used to decide whether or not pruning should be done [59]. As a result, a continuity correction for binomial distribution is used to adjust the error rate [59], [61], [62]. Pessimistic error pruning is performed in a top-down manner. When the internal node is pruned, its child nodes are not pruned.

This ensures a fast pruning process. The continuity correction still yields optimistic error rates [59], [61].

Error-based pruning

Error-based pruning has been implemented in the well-known C4.5 algorithm [61]. It is also used in the M5 model tree induction algorithm [60]. Similarly to pessimistic pruning, error-based pruning does not require a pruning data set. It uses a bottom-up approach. As in pessimistic pruning, the errors are estimated as if the errors have a binomial distribution [61]. It uses the upper bound of the statistical confidence interval for the error counts [61]. Traversing from the bottom to the top of the tree, the pruning technique estimates errors for three different cases and compares them [61]:

- The error rate of the tree when the subtree rooted by node t is not pruned.
- The error rate of the tree when the subtree rooted by node t is pruned.
- The error rate of the tree when subtree rooted by the node t is pruned by replacing the whole subtree with the most frequently used child node of t.

According to the error rate obtained, the procedure either replaces the subtree with a leaf node, leaves the tree as is or replaces the pruned subtree with the most frequently used child node of t [61].

Cost-complexity pruning

Cost-complexity pruning, also referred to as the weakest link pruning, is the pruning method used in the CART system [13], [62]. It consists of two steps [13], [62]:

- In the first step, a sequence of smaller trees, $(T_0, T_1, ..., T_k)$, is induced, where T_0 is the first tree before pruning and T_k is the root node.
- In the second step, the best tree in the sequence is chosen based on its generalization error. The pruned tree with the smallest generalization error is chosen.

Each tree in the sequence is induced such that the succeeding tree is obtained by pruning one or more of the subtrees in the preceding trees. The subtrees that are pruned are those that have the smallest increase in the training data error rate [13], [62]. The error rate is estimated by the following [13]:

$$\alpha = \frac{\varepsilon(pruned(T,t),S) - \varepsilon(T,S)}{|leaves(T)| - |leaves(pruned(T,t))|} \tag{3.11}$$

where $\varepsilon(T,S)$ is the error rate of T, $|leaves(T)|$ represents the number of leaves in T, and sample S and $pruned(T,t)$ is the pruned tree that is obtained when t is replaced with a leaf in T. The cost-complexity pruning method is said to perform well in terms of accuracy. However, the method can be computationally inefficient because a sequence of pruned trees is induced and checked for generalization error. The method is also known to over-prune [25].

Different pruning techniques have been compared in several studies [15], [26], [59]. Although some techniques such as cost-complexity have been shown to over-prune, most of the works concluded that no one pruning technique is always better than all the other pruning techniques.

3.2 Classification tree and regression tree inducers

This section reviews four of the most popular induction algorithms for classification trees and regression trees. The algorithms discussed in this section use the splitting criteria and pruning techniques that were discussed in the previous sections.

3.2.1 ID3

The Iterative Dichotomiser 3 (ID3) decision tree is one of the earliest decision trees to be developed [58]. ID3 uses Hunt's algorithm and does not employ any pruning procedure. It uses gain as the splitting criterion. The stopping criterion used in ID3 requires that every training instance be of the same class or that the information gain should not be greater that than zero [58]. ID3 is known to be the simplest decision tree inducer. One of the disadvantages of the ID3 algorithm is its inability to handle numerical features and

missing values [62]. Because it does not use pruning, ID3 usually overfits the training data when the data contains noise and outliers.

3.2.2 C4.5

C4.5 was developed to tackle the challenges presented by the ID3 algorithm [61]. In fact, the C4.5 algorithm is considered to be evolved from ID3 [61], [62]. The C4.5 algorithm uses the gain ratio as the splitting criterion. The growing stage stops when every training instance belongs to the same value of the target attribute, or when there are no more training instances in a node. After the tree is grown, error-based pruning is performed. C4.5 offers some important enhancements on the ID3 algorithm including the ability to handle missing values and the use of a pruning procedure [61]. C4.5's major challenge is the fact that it keeps all the training data in memory, resulting in poor efficiency [61]. C5.0 was later developed and offers a number of improvements, such as improved efficiency with regard to memory and computational power [61], [62]. Furthermore, C5.0 can be boosted to improve its prediction accuracy [61], [62].

3.2.3 CART

Classification and regression trees (CART) was introduced by Breiman et al. [13]. The first version of CART uses the Gini index as the splitting criterion [13]. This was later replaced by the Twoing criterion because the Gini index is biased towards features with many values [13]. CART uses the cost-complexity pruning technique. An important feature of CART is that it supports both classification tree and regression tree induction. In case of regression, CART splits features based on the minimization of squared errors [13]. The prediction of the regression tree is obtained by obtaining the weighted average of the values in a leaf [13].

3.2.4 CHAID

Chi-squared automatic interaction detection (CHAID) was originally developed for classification and later extended to regression [45]. The original algorithm was designed to handle nominal attributes only. Later on, it incorporated other types of attributes. For

each attribute, CHAID identifies the pair of values that is least significantly different to the target attribute [45]. To estimate the significant differences, a p-value is measured from a statistical test [45]. The statistical test used is dependent on the type of target attribute. For continuous target attributes, an F-test is used. The Pearson chi-squared test is used if the target attribute is nominal [45]. For each selected pair, CHAID evaluates the p-value obtained against a certain merge threshold. If the p-value is greater than the threshold, the values are merged and pairs to be merged are searched for [45]. This process continues until there are no more pairs found. The best attribute for splitting is selected so that the children nodes have homogeneous values of the selected attributes [45]. Splitting continues until a specified threshold is not fulfilled. CHAID uses the following stopping criteria [45]:

- The tree stops growing when the maximum tree depth is reached.
- Reaching the minimum number of instances in a node for being a parent node.
- Reaching the minimum number of instances in a node for being a child node.

As with ID3, CHAID does not perform any pruning.

3.3 Summary

The most important contribution of this chapter was outlining the top-down induction strategy of decision trees in detail. Important steps in inducing decision trees, including splitting criteria, stopping criteria and pruning, were discussed and analyzed. Decision tree induction algorithms related to this work were shortly reviewed. So far, only classification and regression tree induction algorithms were highlighted. The next chapter focuses on model trees.

Chapter 4

Model trees

Within machine learning, most research efforts in decision trees focused on classification trees and regression trees. Another type of decision tree that have received far less attention from the research community is model trees [23]. A model tree can be defined as a decision tree, where the leaf nodes do not have a discrete class value, or a single regression value, but a model of the independent features of the problem. One of the earliest works that explores the concept of using a linear combination of features is CART by Breiman et al. [13], where classification trees use a linear combination of features in the internal nodes only. In the case of regression problems, the authors claimed that the use of linear combination splits in regression is not beneficial [13]. Instead, they suggested that improving the performance of regression trees can be achieved by growing a small tree using the most significant splits and then performing multiple linear regression in the leaf nodes [13]. Quinlan's work later generalized the regression trees in CART by adding regression models in the leaf nodes to improve the accuracy of the prediction [60].

While model trees follow the same induction approach as regression trees, they do not assign a piecewise constant to the leaf nodes, instead they fit functions to their leaf nodes. The prediction for an instance is obtained by traversing down to a leaf and using the corresponding function to compute the final output [8], [33], [60]. Most of the existing model trees in the literature contain linear regression models or kernel regression models in the leaf nodes. Several effective algorithms have been proposed for the induction of model trees. The most popular model tree induction algorithm is

M5, whose leaves implement a general linear function. The M5 model tree induction algorithm was developed by Quinlan [60] and later improved by Wang and Witten [74]. Similar approaches such as the HTL [70], Genetic Program for the Mining of Continuous-Valued Classes (GPMCC) [57], RETIS [44] and the Stepwise Model Tree Induction (SMOTI) [50] were also developed. The M5/M5' induction algorithm, which is used in this study as a base algorithm for inducing a model tree forest, gained its novelty from the fact that it uses simple linear regression models in the leaf nodes. Compared to regression trees and linear regression model, the M5 model tree has proven to be potentially superior in terms of performance [8].

This chapter aims to introduce the M5 tree induction algorithm as well as its extension, M5' tree induction algorithm. Section 4.1 discusses the top induction approach for the M5 algorithm and the M5' algorithm. Section 4.2 gives a short review of similar model tree induction algorithms such as HTL [70] and RETIS [44]. Finally, Section 4.4 provides a summary for this chapter.

4.1 M5 model tree inducer

M5 model trees have been popularly employed for regression tasks. A few studies have extended M5 model trees to classification tasks [31], [54]. Frank et al. has demonstrated that classification problems can be successfully solved by inducing M5' model trees (an extended version of Quinlan's original M5) such that the conditional class probability function of each class is approximated. Using the M5' model tree in classification proved to be more accurate than the state-of-art classifier, C5.0, especially when the features are numeric and binary [31]. The approach, however, proved to be less comprehensible than the C5.0 classifier and less efficient than the C5.0 algorithm when the data sets have many attributes [31]. A similar study conducted by Pal used the M5 model tree for land cover classification [54]. The M5 model tree was compared to a boosted and unboosted univariate decision tree for accuracy and training time. The results from the study revealed that the M5 model tree produced consistently higher classification accuracy with increasing noise in training data when compared to both boosted and unboosted trees [54].

Constructing the M5 model tree involves three important stages [60]. Firstly, a tree is created by employing a top-down approach. The second stage involves finding a model for the data associated with the terminal node and pruning. The final stage involves smoothing. The following sub-sections summarize the M5 model tree construction. Section 4.1.1 discusses the first step of inducing the M5' model tree. Section 4.1.2 focuses the pruning phase and fitting linear models in the leaves of the model tree. Section 4.1.3 discusses the smoothing process.

4.1.1 Constructing the initial tree

Following the standard top-down strategy described in Chapter 3, the M5 model tree induction algorithm recursively partitions the training data into subsets corresponding to the outcome of some test [60]. Quinlan proposes the use of the standard deviation of the target values of the training records in S as the impurity measure [60]. The aim is to maximize the standard deviation reduction (SDR) resulting from testing potential splits at the nodes. The SDR criterion is defined by [60], [74]:

$$SDR = sd\,(S) - \sum_{i} \frac{|S_i|}{|S|} \cdot sd\,(S_i) \tag{4.1}$$

where S_i denotes a subset of instances corresponding to the i_{th} outcome of a test, $sd(S_i)$ is the standard deviation of the target values of observations corresponding to the i_{th} test and $sd(S)$ is the standard deviation of the target values of observations in S. After evaluating all the possible tests, the split that results in the highest expected error reduction is selected after all possible tests have been evaluated [60], [74]. Observe that this is in contrast to CARTs regression tree algorithm which chooses the test that results in the greatest reduction in either absolute deviation or variance of the target values [60], [74]. Malerba et al. [50] argued that choosing optimal splits on the basis of the spread of observations from the mean is not appropriate because the function associated with a leaf node is generally more sophisticated than the sample mean. Thus, the suggestion from the authors is that the optimal split of model trees should be chosen depending on how well the function fits the data.

Growing of the M5 model tree continues until the target values of all the training ex-

amples at the nodes vary only slightly, or the number of the training examples remaining is very small [60].

4.1.2 Leaf models and pruning

During the pruning phase, the M5 algorithm estimates the expected error that will be experienced at each node for unseen data [60], [74]. The expected error is computed by first averaging the absolute difference between the predicted value and the actual value for each of the training observations that reach that node [60], [74]. Because this average will underestimate the expected error on the test data, M5 multiplies this average by multiplication factor, $\left(\frac{n+z}{n-z}\right)$, where n is the number of training observations that reach the node and z is the number of model parameters at that node [60], [74]. After the tree has been grown, the dependent variable is modelled as a function of the independent variables. The linear regression model is calculated using standard regression techniques and is fit in all the internal nodes of the unpruned tree [60], [74]. Instead of using all the features of the data set, the model uses only the features that are tested in the subtree of a node [60], [74]. The resulting regression model is simplified by greedily removing parameters whose effect are small in order to minimize the estimated error calculated using the multiplication factor [60], [74]. Although removing parameters generally increases the average error, it also reduces the multiplication factor. In this way the estimated error is reduced. Once a regression function is in place for each internal node, pruning proceeds in a bottom-up manner from the leaf nodes to the root node. At each internal node, the M5 algorithm compares the estimated error of that node with the estimated error of the subtree below [60], [74]. If the subtree below has a higher estimated error than the node, the subtree will be pruned [60].

4.1.3 Smoothing

A smoothing process is employed in the final stage to avoid inevitable sharp discontinuities between adjacent linear models at the leaf nodes of the pruned tree. The smoothing process is said to enhance the performance of the model tree [60]. Smoothing is particularly important for regression models that are formed from a small number of training

observations. The procedure, as described by Quinlan [60], first uses the regression model in the leaf node to predict the response value, and then filters the predicted value along the same path back to the root. This combines the predicted value in the leaf node with the values predicted by the regression models at each node along the path [60]. The calculation for the predicted value backed up to the next higher node is given by [60]:

$$b' = \frac{nb + dq}{n + d} \tag{4.2}$$

where b' is the prediction backed up to the next node, b is the prediction passed to this node from below, q is the predicted value at this node, n is the number of observations that reach the node below, and d is the smoothing constant with a default value of 15.

An example of the M5 model tree is shown in Figure 4.1, where the leaves contain six regression models.

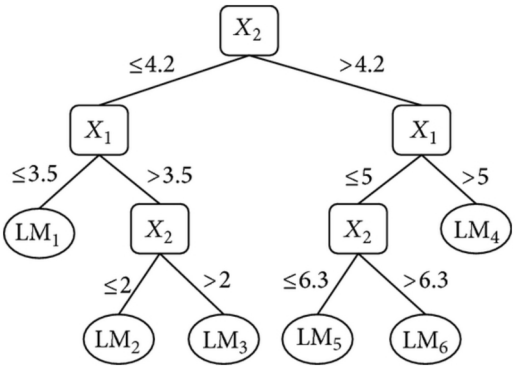


Figure 4.1: Illustration of the M5 model tree with six linear regression models at the leaves.

Quinlan's M5 lacks some important details such as dealing with missing values and categorical variables. Moreover, Quinlan does not clarify the form of linear models used in M5. These challenges later led to the development of the M5' algorithm which provides several improvements to the M5 algorithm [74].

4.2 M5' model tree inducer

The M5' tree induction algorithm was developed by Wang and Witten [74]. It is based on Quinlan's ideas. The M5' algorithm makes five important modifications to the M5 algorithm. Firstly, the form of regression models used in M5 has not been clarified as mentioned in Section 4.1. For a node in the tree with n variables, the regression function at that node is a linear combination of the variables. M5' uses a $n+1$-parameter regression model that has a constant term [74]:

$$w_0 + w_1 a_1 + w_2 a_2 + ... + w_k a_n \tag{4.3}$$

where w_n represents the average number of training observations that reach a node. The multiplication factor used to estimate the expected error, $\left(\frac{n+z}{n-z}\right)$, becomes infinite when a leaf contains one observation, where $n = z = 1$. To overcome this problem, the M5' algorithm constrains the minimum number of training observations in a leaf node to be equal to or greater than two training observations [74].

Secondly, it is stated that M5 grows the tree until the node contains very few training observations or when the training observations vary only slightly [60], [74]. Splitting of a node stops when there are four observations or less, because the leaf node can not contain less than two observations [74]. In addition, M5' stops splitting the node when the standard deviation of the target values of the data points at the node is less than 5% of the standard deviation of the target values of the entire training data set [74].

Thirdly, Quinlan did not clarify which variables are being used in the regression models. While M5 eliminates the terms or attributes whose effect are small, M5' does not eliminate the attributes whose effect are small because they might be useful in higher-level models [74]. Wang and Witten argue that, although leaving these attributes does not influence the error rate in any way, it can produce smaller trees [74].

The fourth modification made to the original M5 algorithm is how the estimated error from the subtree is calculated. Recall that, when pruning a subtree, the estimated expected error for the regression model at that particular node is compared with the estimated expected error of the subtree. In order to compute the estimated expected error of the subtree, the M5' algorithm combines the expected errors from each branch

to form a single overall expected error for the node [74].

The last modification involves dealing with nominal variables and missing values. Prior to growing a tree, the M5' algorithm transforms all the nominal variables into binary variables [74]. This involves calculating the average class value corresponding to each possible value in the enumeration. Then the values in the enumeration are ordered according to the averages of the class values [74]. All splits are binary, involving either continuous variables or synthetic binary variables. So a nominal variable that has k values is replaced by $k-1$ binary variables [74]. To overcome the problem of bias towards multi-valued enumerated variables that do not have predictive power, the M5' algorithm multiplies the SDR by a correction factor β given by [74]:

$$\beta = e^{7\frac{2-k}{n}} \tag{4.4}$$

where k represents the number of values of the original enumerated variable and n is the total number of observations. To process missing values, the M5' algorithm modifies the splitting criterion formula to:

$$SDR = \frac{m}{|S|} \cdot \beta_j \left(sd(S) - \sum_{i \in L,R} \frac{|S_i|}{|S|} \, sd(S_i) \right) \tag{4.5}$$

where m is the number of observations that do not have missing values for that variable, β_j is the correction factor for the original variable corresponding to the synthetic variable, and S_R and S_L are the sets resulting from splitting on this variable.

M5' uses the response value as the surrogate variable to handle missing values during training [74]. In contrast, CART finds another variable to split on in place of the missing value [13]. The variable is chosen on the basis that it has a high correlation with the original variable [13]. Such a technique is complex and time-consuming. Therefore, M5' uses a much simpler technique. Using the response value as the surrogate variable is based on the assumption that this variable is probably correlated with the variable being split [74]. Firstly, the M5' algorithm deals with the observations whose values of the splitting variable are known. These observations are then divided into the sets S_L and S_R according to the test. After evaluating whether the observations in S_L or S_R have the greater average response value, the averages of these two sets are averaged.

An observation for which this variable value is missing is placed into the set S_L or S_R depending on whether the response value of the observation is greater than this overall average or not [74]. If the response value of this observation is greater than the overall average, it goes into the set that has the greater average response value, otherwise it is placed into the set with the smaller average class value [74]. For a test set where the class value is unknown, the missing variable value is replaced by the average of that variable for the training instances that reach the node [74]. The pseudocode for the M5' algorithm is presented in Algorithm 4.1. The two main stages in the algorithm are split and prune. The *sd* function used at the beginning of the algorithm and the beginning of the split stage calculates the standard deviation of the response values of a set of observations. In *split*, *sizeof* gives the number of instances in a set. The SDR is calculated using Equation (4.5). In *prune*, linear regression models are formed at the nodes and the terms are greedily dropped as described earlier.

```
MakeModelTree (instances)
{
    SD = sd(instances)
    FOR each k-values nominal attribute
        convert into k − 1 synthetic binary attributes
    root = newNode
    root.instances = instances
    split(root)
    prune(root)
    printTree(root)
}
split(node)
{
    IF sizeof(node.instance) < 4 or sd(node.instances) < 0.05 ∗ SD
        node.type = LEAF
```

Algorithm 4.1: Pseudocode for M5' model tree induction [74].

```
    ELSE
        node.type = INTERIOR
        FOR each attribute
            FOR all possible split positions of the attribute
                calculate the attribute's SDR
        node.attribute = attribute with mamximum SDR
        split(node.left)
        split(node.right)
}
prune(node)
{
    IF node = INTERIOR then
        prune(node.leftChild)
        prune(node.rightChild)
        node.model = linearRegression(node)
        IFsubtreeError(node) > error(node) then
            node.type = LEAF
}
subtreeError(node)
{
    l = node.left; r = node.right
    IF node = INTERIOR then
        return (sizeof(l.instances) * subtreeError(l) * sizeof(r.instances) * subtreeError(r))
        /sizeof(node.instances)
    ELSE
        return error(node)
}
```

Algorithm 4.1: Pseudocode for M5' model tree induction [74] (Cont.)

Wang and Witten compared the M5 algorithm to the M5' algorithm on six different datasets [74]. The results revealed similar performances by both algorithms on three data sets. On the other three data sets M5' performed better than M5 [74]. The M5' model

tree induction algorithm has been used in a number of applications. Onyari and Ilunga compared the M5 model tree and multi-layer perceptron neural networks in predicting streamflow, and found out that the M5' model tree has a better predictability when compared to multi-layer perceptron neural networks [52]. Another study by Samadi et al. assessed the M5' model tree and CART for the prediction of scour depth [64]. The results of the study showed that the M5' model tree produced more accurate results than the classification and regression trees for the prediction of scour depth [64].

There have been other contributions coming from the machine learning community to develop model trees. The following section reviews some of the popular model tree induction algorithms.

4.3 Other model tree inducers

The HTL model tree proposed by Torgo, employs more sophisticated leaf models such as kernels and local polynomials [70]. Similar to CART, the HTL system minimizes the mean squared error to split the attributes. Although HTL's performance is potentially superior to model trees with linear regression in the leaves, both the construction and usage of HTL is computationally expensive [70]. Karalic argues that the use of the variance of the response variable is not an appropriate measure for impurity when linear regression models are fitted in the leaf nodes [44]. Karalic's argument is based on the fact that it is possible to have large variance in the data even when the relationship between the response variable and the independent variables is linear [44]. This led to the development of the RETIS model tree induction algorithm whose impurity function is the mean square error of the linear regression model in the leaf node. The impurity function I is given by [44] :

$$I = \frac{1}{N\left(t_L\right)} \sum_{i \in t_L} \left(y_i - g_L\left(S_i\right)\right)^2 + \frac{1}{N\left(t_R\right)} \sum_{i \in t_R} \left(y_i - g_R\left(S_i\right)\right)^2 \tag{4.6}$$

where g_L and g_R denote the optimal models for the left and right split respectively.

Using the mean square error of the linear regression models as a splitting criterion builds trees of higher quality. However, the RETIS algorithm is computationally expensive and non-scalable for problems with large datasets [50], [71], [73]. The largest

sample size used for Karalic's experiment is only 300 examples. For continuous variables, a linear system has to be solved for all possible values [42]. Most papers [50], [71], [73] regard RETIS as a non-practical approach. A few approaches have been proposed to address the computational challenges of RETIS. These include SMOTI [50], Scalable Expectation Maximization and Classification based Regression Trees (SECRET) [24], and Smoothed and Unsmoothed Piecewise Polynomial Regression Trees (SUPPORT) [17].

SMOTI is a recent model tree induction algorithm. It induces a tree by using two types of nodes: regression nodes and splitting nodes [50]. The regression nodes create straight-line regressions while the splitting nodes perform partitioning of the features [50]. A multiple linear regression model for each leaf node is constructed stepwise by combining all the straight-line regressions along the same path from the root to the leaf [50]. Thus, the inner regression models have a global effect, contributing to the multiple regression models at the leaf nodes [50]. The straight-line regression at the leaf nodes accounts for local effect [50]. The major disadvantage of the SMOTI model tree is that its split criterion is computationally expensive. This is because it evaluates the mean squared error of the linear models at the leaves as well as all the regression models along the path from the root to the leaf.

In SUPPORT, computational inefficiency is improved by converting the regression problem into a classification problem [17], [50]. As an attempt to replace the exhaustive search used for splitting variables with statistical tests, Chaudhuri et al. [17] proposed the Student's t-test. The basic idea is to fit functional models or simple linear functions in every node of the tree. Then split the data based on the signs of the residuals of the models (positive residuals and negative residuals) [17]. Using the residuals of the models is justified by the priori that, if a fitted model is not satisfactory, then the distributional patterns of the residuals would be able to indicate that there is a lack of fit [17], [50]. Dobra and Gehrke claimed that this justification is not theoretically founded and proposed SECRET [24]. The SECRET algorithm determines two Gaussian clusters with the shape of flat disks by using the expectation-maximization (EM) algorithm on the observation at each node [24]. Once the clusters are determined, an observation is labelled with label 1 if the probability to be in the first cluster exceeds the probability to be in the second cluster. A Gini gain or a similar splitting criterion can then be used

to split the attributes [24]. Then using the least squares linear regression, the models in the leaf nodes are estimated. The advantage of the SECRET algorithm is that it does not solve a large number of linear equations as in RETIS [42]. However, the effectiveness of the SECRET algorithm is dependent on the rationale of the assumption that the data can be split into two groups with the shapes of flat disks.

The GMPCC algorithm proposed by Potgieter and Engelbrecht, uses a genetic programming approach to evolve model trees for continuous-valued classes [57]. The learning process of the genetic programming was improved by using a number of algorithms including, k-means clustering to reduce the training time for the stratified sampling of the training data, candidate non-linear models, and specialized mutation and crossover to evolve model trees such that the models are structurally optimal [57]. The performance of the GMPCC was compared to the performance of other algorithms such as the Cubist (rule-based model extended from the M5' model tree) algorithm and the NeuroLinear algorithm on different data sets [57]. The comparison showed that the performance of the GMPCC method was comparable to that of Cubist and NeuroLinear [57]. Furthermore, the study showed that the GMPCC method was robust and produced number rules that were significantly smaller than smaller than that of Cubist and NeuroLinear [57]. The major disadvantage of the GMPCC method was the slow computational speed of the algorithm. The slow speed of the algorithm is attributed to the nature of recursive procedures that used for fitness evaluation, mutation and crossover on the model trees [57]. To overcome this problem, the authors recommended changing the representation of the model trees to a production system or developing model trees as an array [57].

As with classification trees and regression trees, generating model trees requires a greedy top-down induction strategy that may not yield a global optimal solution.

4.4 Summary

The most significant contribution of this chapter lies in the provision of a detailed background on the M5' algorithm that is used in this study. Section 4.1 discussed and analyzed the M5 algorithm. Section 4.2 discussed the extension of the original M5 algorithm, called the M5' algorithm. Finally, Section 4.3 analyzed some of the model tree

induction algorithms that have been developed in the literature. Chapter 5 focuses on describing the methods that can be used to solve the overfitting problem of model tree induction algorithms.

Chapter 5

Random forest

The main idea of an ensemble method is to combine a set of weak learners, each of which solves the same task, in order to form a much stronger model with improved performance. Ensemble methods such as bagging, random forests and boosting, have been extremely successful in improving decision tree performance [11], [12], [49]. The advantage of ensemble methods with respect to single-induced trees has been reported to be increased accuracy and robustness [11], [12], [49]. The random forest algorithm, proposed by Breiman, is a very powerful ensemble method for building a forest of trees by using a random sample from the original data and selecting random features to split the nodes. To obtain the final prediction, the predictions from the trees are aggregated. In the case of a regression task, the average prediction of the individual trees is used while majority voting is used to aggregate predictions in the case of classification tasks [12]. This chapter briefly reviews the framework of Breiman's random forest. Section 5.1 discusses the random forest algorithm in detail and how the random forest is built. Section 5.2 provides a brief background on the phenomenon of Out-of-Bag error. Section 5.3 reviews the parameters of the random forest model. Section 5.3 provides an overview of the M5' ensemble that have been studied in the literature. Finally, Section 5.6 provides a summary for this chapter.

5.1 Random forest algorithm

The random forest algorithm is given in Algorithm 5.1 and works by growing M trees based on bootstrap samples which are drawn randomly from the original data S. These samples are drawn with replacement (or without replacement) and some data points may be sampled multiple times [12]. Then following the divide-and-conquer procedure, a split is performed at each node over f number of features which are chosen randomly among the F original features, where f is much less than F. Each tree is fully grown without pruning.

Given a dataset S with n total number of data points and F features. Define the total number of trees M to be constructed and the number of variables f to be used for the individual trees such that $f << F$

FOR $i = 1$ to M

Draw a random sample S^i of size n with replacement from S

Using the bootstrap sample S^i as the training data, induce a tree T_i using recursive partitioning:

Grow until a stopping criteria is met

Randomly choose f features from the F features

Find the best binary split from among the f features

Split the node into two child nodes

END

Output the tree $T_f(x)$

END

Make prediction for a given new input x : $\frac{1}{i}\sum_{i=1}^{M} T_i(x)$

Algorithm 5.1: Generic algorithmic for inducing a forest of trees [12], [18].

It is worth noting that there are three important differences between a single-induced tree and a random forest: Firstly, in Breiman's random forest, the splitting criterion is evaluated over a subset of randomly selected features and not over the whole features. The other difference is that the individual trees in the forest are not pruned. Lastly, the trees in the forest are built on bootstrap samples and the samples may contain duplicate

data points.

5.2 Out-Of-Bag Error

An intriguing aspect of random forests is that when a bootstrap sample is drawn from the data, some observations (roughly a third of data) are not used in the tree construction. These observations are denoted by an 'Out-Of-Bag' sample and are used to estimate the generalization error of a random forest and variable importance. The generalization error is referred to as the Out-of-Bag (OOB) error [12]. A useful advantage of the OOB error is that it offers a simple way to tune parameters without the need for a validation set. Unlike cross-validation, the validation step can run in parallel with the model fit in a random forest. In fact, Breiman stated that when the OOB error is estimated, the need for a test set becomes unnecessary [12]. The OOB error is also very useful for determining feature importance in model fitting. The importance of a certain feature is calculated as the average decrease in the prediction error in all trees when the values of that feature is adjusted. The feature which results in the largest increase in the error after being adjusted is considered to be the most important [12].

5.3 Random forest parameters

The random forest algorithm is said to perform quite well with the standard parameter settings [12]. However, there are three parameters that may be optimized to improve the performance [9], [34]:

- The number of trees in the forest
- The number of randomly selected features chosen at each node
- The smallest node size for splitting

For regression problems, the standard recommended default value for the number of randomly selected features is $f = F/3$, while for classification tasks, the standard value is $f = F^{1/2}$ [12]. The effect of this parameter is investigated by Genuer et al. who argued

that the default value of the number of the randomly selected features is either optimal or too small. As a result, Geneur et al. suggested that the value must be as large as possible [34]. A large number of trees is reported to result in variance reduction [12]. Thus, the forest is likely to give more accurate predictions when the number of trees is increased. Furthermore, increasing the number of trees does not lead to overfitting. For a small number of trees, the out-of-bag error can be inaccurate and unstable. However, as the number of trees increases, Breiman proved that the generalization error of random forests converges to a minimum [12]. In practice, this means that the value for the number of trees can be chosen as large as desired without overfitting. However, the computational cost for growing a forest of trees increases with the number of trees [66]. Therefore, there should be a trade-off between accuracy and computational cost. Another interesting and related work was done by Oshiro et al. [53] who argued that sometimes a larger number of trees only lead to greater computational cost with no significant performance gain. Based on the experiments, Oshiro et al. suggested a range between 64 and 128 trees in a forest. With these numbers of trees, Oshiro et al. claim that it is possible to get a very good balance between computational time, accuracy, and memory [53].

5.4 Advantages and disadvantages of random forest

What has greatly contributed to the increased popularity of random forests is the fact that they yield high accuracy on most prediction problems and reduce variance in decision tree models. They are versatile enough to be applied to both classification and regression tasks [10]. The algorithm is also recognized for its ability to handle data with high-dimensional feature spaces. Table 5.1 summarizes the strengths and the weaknesses of the random forest model.

Table 5.1: The advantages and disadvantages of random forest.

Advantages	**Disadvantages**
Can be applied to both classification and regression problems and has few parameters to tune.	Unlike decision trees, random forest is not easily interpretable. Similar to neural networks, random forest is considered a black-box model.
Random forest generalizes new data well. It does not overfit the training data.	Compared to decision trees, random forest is computationally expensive.
Can handle noisy or missing data.	
Can be used to determine variable importance.	
It can run efficiently on data with an extremely large number of variables.	

5.5 M5/M5' ensembles

Ensemble methods have been originally developed for classification and regression trees. Ensembles of M5' model trees have not been widely studied. In the literature, very few authors such as Sattarri et al. [65], Aleksovski [3] and Pfahringer [56], have studied ensembles of model trees. Pfahringer's semi-random model tree ensembles use bagging, where the splits are based on the median of some attribute to avoid skewed trees, i.e. leaves that have a large difference in the number of training data points [56]. Sattarri et al. used four ensembles of M5' model trees, including stochastic gradient boosting, bagging, random sub-space, and a rotation forest, to predict sodium adsorption ratio. The predictions by the different ensemble methods were compared with that of the M5' model tree [65]. With the exception of random sub-space, the ensemble methods were shown to perform better than the M5' model tree algorithm. Also, reducing the attributes of the data sets was shown to improve the performance of the M5' model tree algorithm and the ensemble methods used [65]. Satarri et al. reported increased computational

complexity and interpretability as the major challenges in building ensembles of model trees [65].

Aleksovski investigated crisp and fuzzy model tree (modified model trees) ensembles for modeling dynamic systems [3]. Aleksovski's ensemble construction procedure was based on the popular bagging and random forest techniques containing 100 trees. Following the bagging and random forest procedure, bootstrap samples are created and fuzzied model trees are induced. After the ensemble is built, it is optimized by eliminating trees that do not contribute to the performance of the ensemble [3]. To evaluate a tree's contribution to the accuracy of the ensemble, the error of the ensemble without the tree is compared to the error of the ensemble with the tree [3]. Eventually, the ensemble selection procedure is stopped when removing a tree results in no further improvement [3]. After the ensemble selection procedure, the final prediction of the model tree ensemble is obtained by computing the weighted average predictions of the individual trees [3]. In this study, the results indicated that a forest of crisp and fuzzy M5' model trees performed better a single M5' model tree. In a similar study, Aleksovski et al. [4] introduced a novel ensemble method which includes bagging and a random forest for noise-tolerant system identification. The results showed that the fuzzy model tree ensembles performed better than both regression tree ensembles and single fuzzy model trees. In their observations, the optimal number of trees was 50 and increasing this number showed no improvement in the accuracy of both bagging and the random forest [4]. On the other hand, the size of the random features selected for splitting, had a significant effect on the performance of the random forest [4].

5.6 Summary

The aim of this chapter was to provide background on random forests. The chapter provided and analyzed the generic algorithm for random forests. Furthermore, the parameters of random forests were discussed. The chapter also reviewed the existing literature on M5' ensembles. The next chapter covers all aspects relating to the experimental design including experimental methodology, the data sets used, and an analysis of the results.

Chapter 6

Experimental design and analysis

This chapter discusses the experimental procedure and analyzes the performance of the M5' single model trees and the M5' model tree forests on various data sets obtained from the UCI machine learning repository [22] and KEEL (Knowledge Extraction based on Evolutionary Learning) [2] data set repository. This chapter will try to provide answers to the following questions:

- Do M5' model tree forests perform better than single-induced M5' model trees?
- What is the influence of the number of trees and number of randomly selected features on the performance of the M5' forest?

Section 6.1 presents the data sets used in the experiment. The empirical process is discussed in Section 6.2. Section 6.3 presents the experimental results and the analysis of the results. The chapter is finally summarized in Section 6.4.

6.1 Data sets and pre-processing

This section describes the data sets used in the experiments and the pre-processing procedure of the data sets.

The M5' forest was evaluated on a number of machine learning data sets. These data sets have a continuous target variable and contain varying numbers of numeric and nominal variables, and instances. They have been commonly used as benchmarks in

studies evaluating model trees and regression trees. A description of the data sets is given in Table 6.1.

Table 6.1: Overview of the data sets used in this study (Attributes: C denotes Continuous, N denotes Nominal).

Data set	**Instances**	**Attributes**	**Prediction task**
Abalone	4177	7 (8C,1N)	Age of abalone
Auto-MPG	396	8 (4C,4N)	Car fuel consumption
Bank8FM	8192	8 (8C)	Fraction of customers who leave the bank because of long queues
Boston housing	506	13 (12C,1N)	Median values of homes in Boston
Delta-elevators	9517	6 (6C)	Variation in the action taken on the aircraft elevators
Elevators	16599	18 (18C)	Action taken on the elevators of the aircraft
Machine-CPU	209	6 (6C)	CPU relative performance
Mortgage	1049	15 (15C)	30 year-conventional mortgage rate
Servo	167	4 (2C,2N)	Response time of a servo mechanism
Wine quality	4898	11 (11C)	Wine quality based on physicochemical tests

The pre-processing procedure involved dealing with the missing values in the data sets and removing non-predictive variables from the data sets. The missing values in the data sets were replaced with means or the medians of the training data sets. For data sets that contain outliers, the median was used for imputing missing values. All the categorical variables in the data sets were converted into numeric variables.

6.2 Empirical process

This section describes the experimental procedure used in this study. Section 6.2.1 discusses the sampling procedure, the algorithms and the parameter settings used in the experiments. Section 6.2.2 discusses the evaluation of the algorithms.

6.2.1 Experimental procedure

The M5' model trees and the M5' model tree forests were implemented using the M5PrimeLab toolbox [43]. The M5PrimeLab is an Octave/Matlab toolbox for building M5' model trees and M5' regression trees as well as ensembles of M5' model trees. The toolbox implements the M5' trees according to the Wang and Witten method [74], and the random forest according to Breiman's method [12].

In order to evaluate the performance of the algorithms, each of the data sets was partitioned by means of a repeated 10-fold cross-validation. The 10-fold cross validation method splits the data into 10 parts of approximately equal sizes. Of the 10 sub-samples, a single sub-sample is used as the validation set for testing the model, and the remaining sub-samples are used as the training data set.

A list of the optimal parameter settings for each of the algorithms and data set is reported in Table 6.2. The M5' model trees were grown to their full depth using the default parameters and then optimized by pruning. For the M5' model tree forest, the two important parameters considered were the number of trees (M) and the randomly selected features (f). To determine the optimal combination of these two parameters, several different parameter values were tested and the combination that gives the least error was chosen.

6.2.2 Performance measure

The experimental analysis evaluates the performance of the models learned using the predictive performance of the algorithms as a measure of performance. The predictive performance of the models were measured in terms of the Root Mean-Squared Error (RMSE), calculated as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{6.1}$$

To evaluate the statistical significance of the differences observed in the experiments for each data set, the non-parametric Mann-Whitney U test was used. The Mann-Whitney test is commonly used when the assumptions of the normal distribution and homogenity of variance can not be made. The test was applied to the errors obtained from the cross-validation method, that is, the performance from each of the folds. The null hypothesis, which states that the two populations are not significantly different, is tested against the two-sided alternative hypothesis which states that the two populations are different. In all experiments reported in this empirical study, the significance level used in the test is set at 0.05. Thus, the null hypothesis is rejected when the p-value is less than the significance level of 0.05.

Table 6.2: Parameters considered for the algorithms for each data set. M and f denote the number of trees and the number of randomly selected features respectively.

Data set	**Algorithm**	**Parameter**
Abalone	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 300$ $f = 2$
Auto-mpg	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 100$ $f = 2$
Bank8FM	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 250$ $f = 4$
Boston housing	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 50$ $f = 5$

Table 6.2: Parameters considered for the algorithms for each data set. M and f denote the number of trees and the number of randomly selected features respectively (Cont.).

Data set	Algorithm	Parameter
Delta elevators	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 200$ $f = 2$
Elevators	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 100$ $f = 6$
Machine CPU	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 200$ $f = 4$
Mortgage	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 200$ $f = 3$
Servo	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 100$ $f = 3$
Wine quality	M5'	Pruning=true Minimum leaf size=2
	M5' forest	$M = 100$ $f = 3$

6.3 Experimental results and empirical analysis

This section presents the results of the experiments described in Section 6.2. The analysis of the results is also provided. First, Section 6.3.1 compares the performance of the M5' model trees to the performance of the M5' forests on the benchmark data sets. Then, Section 6.3.2 evaluates the performance of the M5' forest with reference to the number of trees and the number of randomly selected features for splitting the node.

6.3.1 Comparing the single-induced M5' model tree to the M5' forest

This subsection compares the performance of the M5' model tree forest to the performance of the single-induced M5' model tree.

Table 6.3 reports the average Root Mean Square Error (RMSE) values obtained from the cross validation method and the p-values for each of the ten data sets. The statistically significant values (p-value < 0.05) are presented in bold and the standard deviation values of the errors are within parentheses. All statistically significant values are favorable to the M5' forest.

Table 6.3: M5' forest vs. M5' model tree: Results of the Mann-Whitney U test on the errors of the algorithms.

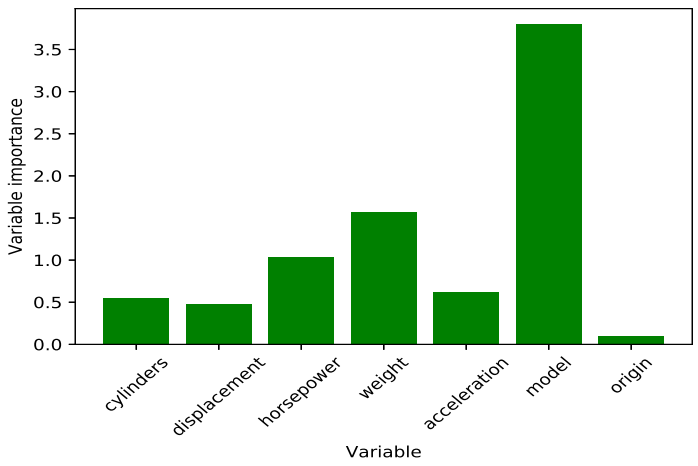
Data set	**RMSE**		p**-value**
	M5' model tree	**M5' model tree forest**	
Abalone	2.16988 (0.15939)	2.08474 (0.12392)	**0.029 75**
Auto-MPG	2.81596 (0.4841)	2.57652 (0.6078)	$6.676\,26 \times 10^{-2}$
Bank8FM	0.03152 (0.00090)	0.02987 (0.00111)	$\mathbf{1.024\,11 \times 10^{-7}}$
Boston housing	3.61008 (1.06155)	3.34285 (0.80645)	$4.420\,21 \times 10^{-1}$
Delta-elevators	0.00145 (0.00004)	0.00140 (0.00004)	$\mathbf{6.728\,30 \times 10^{-4}}$
Elevators	0.00236 (0.00008)	0.00221 (0.00008)	$\mathbf{1.204\,42 \times 10^{-7}}$

Table 6.3: M5' forest vs. M5' model tree: Results of the Mann-Whitney U test on the errors of the algorithms (Cont.).

Data set	**RMSE**		
	M5' model tree	**M5' model tree forest**	p**-value**
Machine-CPU	50.28930 (24.53751)	46.75612 (25.11360)	$\mathbf{0.428\,12 \times 10^{-1}}$
Mortgage	0.11855 (0.02644)	0.08181 (0.01148)	$\mathbf{1.477\,23 \times 10^{-9}}$
Servo	0.57640 (0.26802)	0.52847 (0.23797)	$3.830\,55 \times 10^{-1}$
Wine quality	0.69770 (0.02552)	0.65983 (0.02160)	$\mathbf{1.334\,76 \times 10^{-6}}$

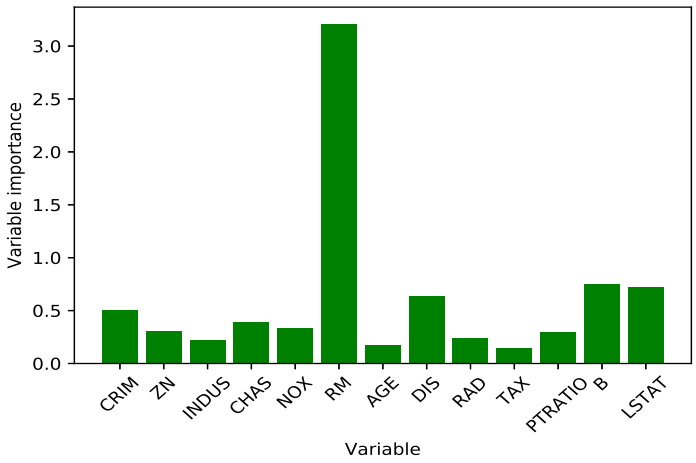
Table 6.3 reveals that the M5' model tree forests always produced smaller errors than the single M5' model trees for all the data sets. This observation is expected and is in accordance with the literature that random forests improve the predictive performance of decision trees. The significant values show that for 7 out the 10 data sets, the M5' forest performance was significantly better than the M5' model tree performance. In particular, the M5' forest did not provide significant improvement to the M5' model tree on three data sets, namely Boston housing, Auto-mpg and Servo data sets, even though the average RMSE of the M5' forest is better than that of the M5' model tree. All of these three data sets have a combination of continuous and categorical variables. With the exception of the Abalone data set, the 7 data sets for which the M5' forest performed significantly better than the M5' model tree contain continuous variables. The Servo data set contains two nominal variables and two continuous variables, the Boston housing data set contains one binary variable and twelve continuous variables, and the Auto-mpg data set contains four nominal variables and and four continuous variables. However, it can not be deduced that the presence of categorical variables in the data sets affected the performance of the M5' model tree forest or that there

was a preference of continuous variables over categorical variables during splitting. As discussed in Chapter 4, M5' trees can handle both continuous and categorical variables. Furthermore, prior to growing the M5' tree, all the nominal variables are converted into binary variables. However, it was interesting to investigate the contribution of the variables to the prediction of the trees to get an overview of how the contribution of the categorical variables compares to the contribution of the continuous variables in the tree prediction. A variable importance score was estimated by the random forest algorithm on the three data sets. The importance score assigned to a variable indicates how often that variable was chosen, and how much of an effect it had in minimizing the loss function when it was chosen. Figure 6.1 shows the variable importance measures for the Boston housing, Servo and Auto-mpg data sets for the trees generated. Larger scores indicate bigger importance of the corresponding variable. Figure 6.1 indicates that the importance of the categorical variable (CHAS) in the Boston housing data set was comparable to the other continuous variables. Similarly, the variable importance scores of the nominal variables (motor and screw) in the Servo data set appeared to comparable to the most important variable, pgain. Therefore, there is no evidence that the M5' algorithm and its ensembles favours continuous variables over categorical variables when splitting.

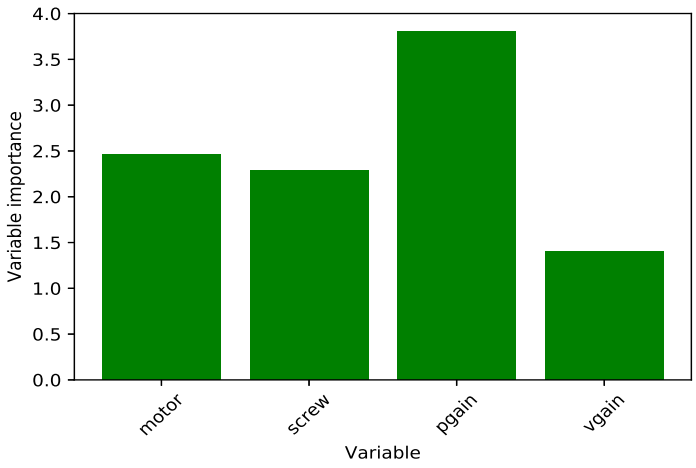


(a) Auto MPG

Figure 6.1: Random forest variable importance estimated for (a) Auto MPG, (b) Boston housing and (c) Servo data sets.



(b) Boston



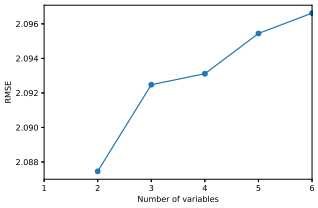
(c) Servo

Figure 6.1: Random forest variable importance estimated for (a) Auto MPG, (b) Boston housing and (c) Servo data sets (Cont.).

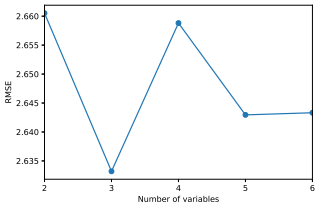
Regarding the variance of the algorithms, Table 6.3 highlights that the M5' forest had a lower standard deviation than the single-induced M5' model trees on 7 of the 10 data sets. A lower value of standard deviation indicates that the model prediction possesses less variability with different subsets of training data. The M5' model trees have generally higher variances on the data sets. This was expected since decision trees are known to be high-variance models and thus overfit the training data [62].

6.3.2 The effect of the M5' forest parameters

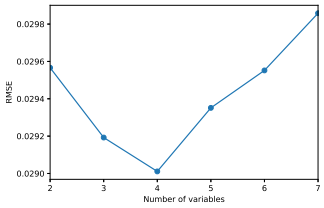
As stated in Section 5.3, Breiman points out two parameters that significantly affect the performance of a random forest, that is, the number of trees (M) and the number of randomly selected variables at each node (f). This section analyzes and discusses the effect of these two important parameters on the performance of the M5' model tree forest. To determine the relationship between the performance of the M5' forest and the parameters, the OOB error rate was examined while one parameter was held constant and the other parameter was varied. Figure 6.2 shows the influence of varying the number of variables on the performance. The number of trees was held constant at 100 trees. Based on the results, the error rates did not change dramatically when the number of variables was varied. The random forests for the Abalone, Delta-elevators, Servo, Boston housing and Bank8FM, showed optimal performance when the number of variables was set at the default setting ($f/3$). For most data sets, increasing the number of variables at each split did not decrease the error rate. It can be observed that using a very large number of variables increased the error rates. In the original random forest paper [12], Breiman introduced strength and correlation as the key features that influence the performance of a random forest. Breiman noted that the error rate of a forest is dependent on the correlation among the trees and the strength of the individual trees. Increasing the correlation leads to an increase in the error rate of the forest. Furthermore, increasing the strength of the individual trees decreases the error rate of the forest. Increasing the number of variables increases both the strength and correlation [12]. If too many variables are used, the correlation is increased, thus resulting in an increased error. Also, a very small number of variables may lead to an increased error. An optimal number of trees exists between this range, where an increase in the number of variables increases the correlation among trees, but also decreases the error. It can be said that, in practice, the optimal number of variables selected for a split depends on the problem and the data set.



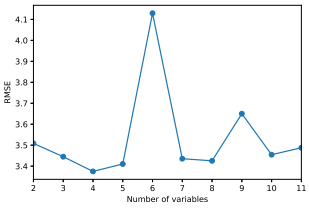
(a) Abalone



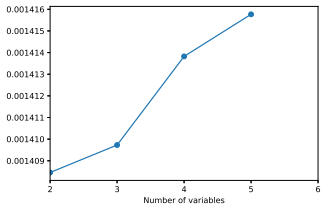
(b) Auto-MPG



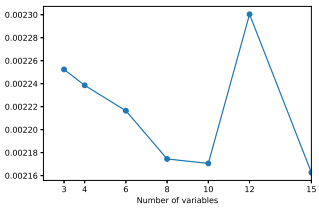
(c) Bank8FM



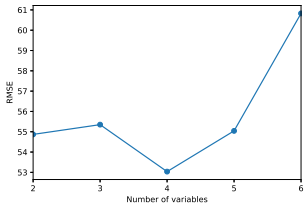
(d) Boston



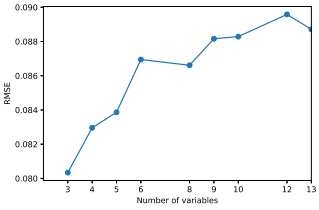
(e) Delta-elevators



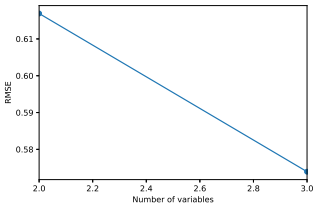
(f) Elevators



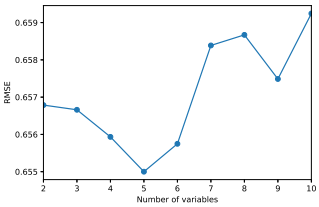
(g) Machine-cpu



(h) Mortgage



(i) Servo

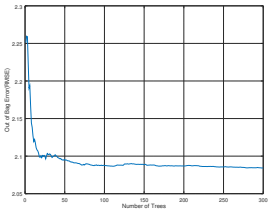


(j) Wine quality

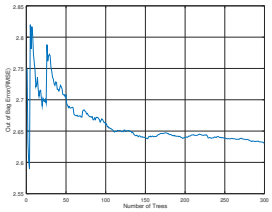
Figure 6.2: Effect of the number of random split variables at each node on the OOB error for all the data sets.

The effect of the number of trees on the performance of the M5' forest is depicted in Figure 6.3. These graphs show the evolution of the OOB error rate as a function of the number of trees when the number of the variables is held constant at the default value ($f/3$). For all the data sets, the OOB error decreased rapidly until it converged to a data-specific value. This pattern is typical of what is observed in classification tree and regression tree literature [12]. The OOB error rate for a very small number of trees was divergent before stabilizing in convergence as more trees are added. For most larger data sets (Bank8FM, Delta-elevators, Elevators Abalone) the OOB error rate stabilized into convergence much quicker at about 50 trees. Smaller data sets took longer to reach stabilization. A detailed look at the OOB error for the Machine-cpu data set reveals that the error varied a lot between 1 and 100 trees. After 100 trees, it is clear that the error became steady and reached convergence. For the Boston housing data set, the OOB slightly increased again after reaching its lowest value, leading to a value at 300 trees that is 0.1 greater than the lowest value of the curve for $M \in [10, 275]$. This is typical for smaller data sets where a few data points can have a high impact on the OOB error rate.

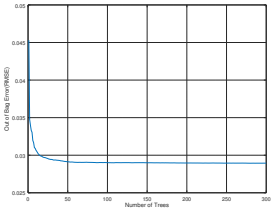
Based on the results, it can be said that increasing the number of trees yielded improvement in the OOB error of the random forests, however, above a certain number of trees the improvement became insubstantial. The biggest performance gain in the OOB error was seen at the first 100 trees for most data sets. Thus, the improvement decreases as the number of trees increases. After a certain number of trees the benefit in the predictive performance from adding more trees will be lower than the cost of computational time for training more trees. Therefore, the number of trees should be chosen as a compromise between performance and computational time.



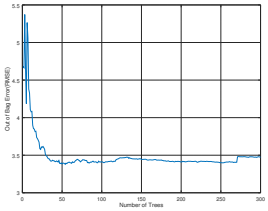
(a) Abalone



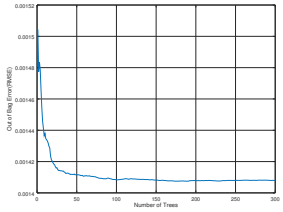
(b) Auto-mpg



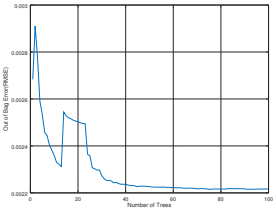
(c) Bank8FM



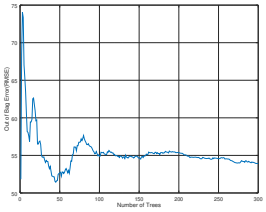
(d) Boston



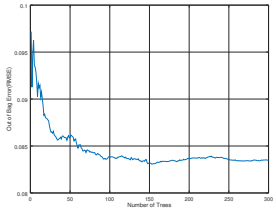
(e) Delta-elevators



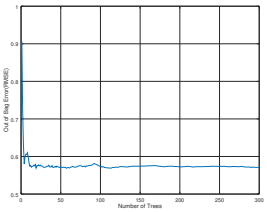
(f) Elevators



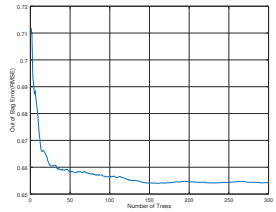
(g) Machine-cpu



(h) Mortgage



(i) Servo



(j) Wine quality

Figure 6.3: Effect of the number of trees on the OOB error for all the data sets.

6.4 Summary

This chapter examined and discussed the experimental procedure and the experimental results for this dissertation. Section 6.1 presented the data sets used in the experiment. The experimental procedure was provided in Section 6.2. Section 6.3 provided an analysis for the experimental results where the single-induced M5' model tree performance was compared to the proposed M5' forest performance. Section 6.3 further analyzed the M5' forest performance with regards to the number of variables selected for a split at each node and the number of trees in the forest. The results showed that for 7 out the 10 data sets, the M5' forest performance was significantly better than the M5' model tree performance. The OOB error decreased with the increasing number of trees until converging to a data-specific value.

Chapter 7

Conclusions

This chapter summarizes the main findings of this study and suggests future work.

7.1 Summary of conclusions

The main objective of this study was to develop an M5' model tree forest and to compare its performance with that of the single-induced M5' model tree. The second objective was to analyze the M5' model tree forest performance with respect to the number of trees and the number of variables randomly selected for splitting at each node. To achieve these objectives, important steps that led to the development of the empirical procedure were taken.

The research work commenced in Chapter 2 with a discussion of the background of decision tree learners in the machine learning domain. Important concepts were defined in order to build a solid base of background on decision trees. Chapter 3 discussed the divide and conquer strategy including the splitting and stopping criteria. The chapter also introduced the phenomenon of overfitting which represent a major problem in decision trees. Chapter 4 was mainly dedicated to the M5' algorithm. A detailed background on the procedure of growing the M5' model tree as well as the overfitting problem was covered. The knowledge was vital in selecting parameters for the experimental procedure in Chapter 5. The random forest algorithm was thereafter explored as a solution to overcoming the overfitting problem in M5' model trees. Parameters affecting the

performance of the random forest algorithm were also discussed.

The core of this study took the form of experiments and analysis. The experimental procedure that was conducted in this work was presented in Chapter 6 and comprised two parts. The first part of the experiment analyzed the performance of the M5' forest when compared to the single-induced M5' model tree on ten benchmark data sets. For each algorithm, the parameters were optimized over each of the ten benchmark data sets. The performance of each algorithm was evaluated by means of repeated cross-validation. To assess the statistical significance of the differences observed in the performance of the algorithm for each data set, a non-parametric Mann-Whitney U test was conducted.

The major findings from the first part of the experiment can be summarized as follows. The M5' forest performed significantly better than the single-induced M5' model tree on seven of the ten data sets. The M5' forest performed insignificantly better on the other three data sets. Overall, the M5' forest improved performance of the single M5' model tree. These results are very promising and prove the potential power of random forests in model trees.

The second part of the experiment analyzed the M5' forest performance with respect to the number of trees in the forest and the number of randomly selected variables for splitting each node of the tree. The experiment involved changing one parameter while keeping the other constant. The main findings from this part were that the addition of more trees to the forest improved the performance of the forest. However, the improvement in the performance of the forest decreased as the number of trees increased while the computational time increased.

It is recommended that the number of trees be tuned, instead of simply choosing a very large number of trees. Another finding was that the M5' forest performed well at the default number of variables randomly selected for splitting. Choosing a very large number relative to the total number of variable for this parameter did not necessarily improve the performance of the M5' forest.

7.2 Future work

Several ideas for future research were identified. A brief overview is listed below.

Application of M5' forest to larger data sets
The first direction for further work considers extending the use M5' forest to larger data sets with more variables and observations. The application of M5' model trees in the literature has been limited to smaller data sets. Since random forests can be computationally expensive, exploring the M5' model with larger data sets may prompt a research in improving the computational time of the M5' forest algorithm. Building trees in parallel can be considered in this case.

Ensemble of other model trees
This study limited the base learner of the random forest to M5' model trees. Many studies involving ensembles of model trees have been limited to M5' model trees. An investigation into developing ensembles of other model trees such as HTL and SMOTI would be interesting.

Investigation of bagging size
Breiman's random forest algorithm draws a random bootstrap sample of size n from the original data set of size n. Investigation into the effect of bootstrap sample size on the performance of model tree forests may make a worthwhile contribution to the research community.

Application of M5' model tree forests to data streams
Today, many data sources, such as sensor networks, social networks and financial markets, produce a huge amount of data at high speeds. These data streams introduce new challenges and require that analysis take place in real-time. Application of M5' model trees and M5' model tree forests to data streams could be an interesting research focus for future work.

Bibliography

[1] A. V. Aho, J. E. Hopcrof, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Massachusetts, California, 1974.

[2] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garcia, L. Sanchez, and F. Herrra. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011.

[3] D. Aleksovski. *Tree Ensembles for Discrete-Time Modeling of Non-Linear Dynamic Systems*. Doctor of philosophy thesis, Jozef Stefan International Postgraduate School, Slovenia, 2014.

[4] D. Aleksovski, J. Kocijan, and S. Dzeroski. Model-tree ensembles for noise-tolerant system identification. *Advanced Engineering Informatics*, 29(1):1–15, 2015.

[5] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, Cambridge, 2010. Available online: [ISBN:026201243X,9780262012430].

[6] A. Appice and S. Dzeroski. Stepwise induction of multi-target model trees. In J. Kok, J. N.and Koronacki, R. L. Mantaras, S. Matwin, D. Mladenic, and A. Skowron, editors, *Proceedings of the 18th European Conference on Machine Learning*, pages 502–509, Berlin, 2007. Springer.

[7] R. C. Barros, A. C. P. L. F. De Carvalho, and A. A. Freitas. *Automatic Design of Decision-Tree Induction Algorithms*. Springer, Switzerland, 2015.

[8] R. C. Barros, D. D. Ruiz, and M. P. Basgalupp. Evolutionary model trees for handling continuous classes in machine learning. *Information Sciences–Informatics and Computer Science, Intelligent Systems, Applications: An International Journal*, 181(5):954–971, March 2011.

[9] S. Bernard, L. Heutte, and S. Adam. A new random forest induction method. In *Advanced Intelligent Computing Theories and Applications with Aspects of Artificial Intelligence*, pages 430–437, Berlin, 2008. Springer.

[10] G. Biau and E. Scornet. A random forest guided tour. *TEST: An official journal of the Spanish Society of Statistics and Operations Research*, 25(2):197–227, 2016.

[11] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[12] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[13] L. Breiman, F. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Taylor Francis, Monterey, California, 1984.

[14] A. Buja and Y. S. Lee. Data mining criteria for tree-based regression and classification. In *KDD '01 Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 27–36. ACM, 2001.

[15] W. Buntine. *A theory of learning classification rules*. PhD thesis, School of Computing Science, University of Technology, Sydney, University of Technology, Sydney, November 1992. Available online: [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.5614&rep=rep1&type=pdf] (Accessed: 21 July 2018).

[16] P. Chaudhuri. Asymptotic consistency of median regression trees. *Journal of Statistical Planning and Inference*, 91:229–238, 2000.

[17] P. Chaudhuri, M. C. Huang, W. Y. Loh, and R. Yao. Piecewise-polynomial regression trees. *Statistica Sinica*, 4(1):143–167, January 1994.

[18] A. Cutler, D. R. Cutler, and J. R. Stevens. Random forests induction. In *Ensemble Machine Learning*, pages 157–176. Springer, 2012.

[19] R. L. De Mantaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6(1):81–92, 1991.

[20] B. de Ville. *Decision Trees for Business Intelligence and Data Mining: Using SAS Enterprise Miner*. SAS Institute Inc., Cary, North Carolina, 2016.

[21] G. De'ath. Multivariate regression trees: A new technique for modeling species-environment relationships. *Ecological Society of America*, 83(4):1105–1117, 2002.

[22] D. Dheeru and E. K. Taniskidou. UCI machine learning repository, 2017. Department of Information and Computer Sciences, University of California, Irvine, United States of America. Available online: [http://www.ics.uci.edu/~mlearn/MLRepository.html] (Accessed: 13 April 2018).

[23] A. Dobra. *Classification and Regression Tree Construction*. Thesis proposal, Cornell University, Cornell University, Department of Computer Science, Ithaca, New York, November 2002.

[24] A. Dobra and J. Gehrke. Secret: A scalable linear regression tree algorithm. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 481–487, New York, 2002. ACM.

[25] F. Esposito, D. Malerba, and G. Semeraro. A further study of pruning methods in decision tree induction. In *Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics*, pages 211–218, 1995.

[26] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transanctions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[27] U. M. Fayyad and K. B. Irani. The attribute selection problem in decision tree generation. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 104–110. AAAI Press/MIT Press, 1992.

[28] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54, 1996.

[29] C. Ferri, P. Flach, and J. Hernandez-Orallo. Learning decision trees using the area under the roc curve. In *Proceedings of the 19th International Conference on Machine Learning*, pages 139–146. AAAI Press/MIT Press, July 2002.

[30] E. Frank. *Pruning Decision Trees and Lists*. Doctor of philosophy thesis, University of Waikato, New Zealand, 2000.

[31] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten. Using model trees for classification. *Machine Learning*, 32:63–76, 1998.

[32] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, 13(3):57–70, 1992.

[33] J. Gama. Functional trees. *Machine Learning*, 55(3):219–250, 2004.

[34] R. Genuer, J. M. Poggi, and C. Tuleau-Malot. Variable selection using random forests. *Pattern Recognition Letters*, 31(14):2225–2236, 2010.

[35] K. Grabczewski and N. Jankowski. Feature selection with decision tree criterion. In *Proceedings of the 5th International Conference on Hybrid Intelligent Systems*, pages 212–217, Rio de Janeiro, Brazil, 2005. IEEE.

[36] B. Gupta, A. Rwat, A. Jain, A. Arora, and N. Dhami. Analysis of various decision tree algorithms for classification in data mining. *International Journal of Computer Applications*, 163(8):15–19, 2017.

[37] N. Gupta, S. Nickolas, and A. V. Reddy. Feature selection using decision tree induction in class level metrics dataset for software defect predictions. In *Proceedings of the World Congress on Engineering and Computer Science*, pages 124–129, San Francisco, USA, October 2010.

[38] T. Hancock, T. Jiang, M. Li, and J. Tromp. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(40):114–122, 1996.

[39] C. Hartmann, P. Varshney, K. Mehrotra, and C. Gerberich. Application of information theory to the construction of efficient decision trees. *IEEE Transactions on Information Theory*, 28(4):565–575, July 1982.

[40] E. B. Hunt, J. Marin, and Stone P. J. *Experiments in Induction*. Academic Press, Oxford, England, 1966.

[41] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1991.

[42] E. Ikonomovska. *Algorithms for Learning Regression Trees and Ensembles on Evolving Data Streams*. PhD thesis, Jozef Stefan International Postgraduate School, Ljubljana, Slovenia, October 2012.

[43] G. Jekabsons. M5PrimeLab: M5' regression tree and model tree toolbox for matlab/octave ver. 1.7.0. Technical report, Faculty of Computer Science and Information Technology Riga Technical University, Riga, Latvia, 2010.

[44] A. Karalic. Employing linear regression in regression tree leaves. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 440–441, New York, 1992. John Wiley & Sons.

[45] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119–127, 1980.

[46] M. Kearns and Y. Mansour. A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. In *Machine Learning: Proceedings of the 15th International Conference*, pages 269–277. Morgan Kaufmann Publishers, 1998.

[47] S. B. Kotsiantis. Decision trees: A recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013. Available online: [https://doi.org/10.1007/s10462-011-9272-4].

[48] V. E. Lee, L. Liu, and R. Jin. Decision trees: Theory and algorithms. In *Data Classification: Algorithms and Application*, pages 87–119, 2014.

[49] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.

[50] D. Malerba, F. Esposito, M. Ceci, and A. Appice. Top-down induction of model trees with regression and splitting nodes. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, 26(5):612–625, May 2004.

[51] O. J. Murphy and R. L. McCraw. Designing storage efficient decision trees. *IEEE Transanctions on Computers*, 40(3):315–320, March 1991.

[52] E. K. Onyari and F. M. Ilunga. Application of MLP neural network and M5P model tree in predicting streamflow: A case study of luvuvhu catchment, south africa. *International Journal of Innovation, Management and Technology*, 4(1):11–15, February 2013.

[53] T. M. Oshiro, S. P. Pedro, and J. A. Baranauskas. How many trees in a random forest? In *Proceedings of the 8th International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 154–168, Berlin, Germany, 2012. Springer.

[54] M. Pal. M5 model tree for land cover classification. *International Journal of Remote Sensing*, 27(4):825–831, 2006. Available online: [DOI:10.1080/01431160500256531].

[55] K. Pattipati and M. Alexandridis. Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Transactions on Sytems, Man and Cybernetics*, 2(4):872–887, 1990.

[56] B. Pfahringer. Semi-random model tree ensembles: An effective and scalable regression method. In *Proceedings of the 24th Australisian Conference on Artificial Intelligence*, pages 231–240, June 2011.

[57] G. Potgieter and A. P. Engelbrecht. Evolving model trees for mining data sets with continuous-valued classes. *Expert Systems with Applications*, 35:1513–1532, 2008.

[58] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[59] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies - Special Issue: Knowledge Acquisition for Knowledge-based Systems. Part 5*, 27(3):221–234, September 1987.

[60] J. R Quinlan. Learning with continuous classes. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, pages 343–348. World Scientific, 1992.

[61] J. R. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, San Fransico, 1993. Available online: [ISBN:1-55860-238-0].

[62] L. Rokach and O. Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific Publishing, Singapore, April 2014.

[63] R. S. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, June 1992.

[64] M. Samadi and E. Jabbari. Assessment of M5' model tree and classification and regression trees for prediction of scour depth below free overfall spillways. *Neural Computing and Applications*, 24(2):357–366, July 2014.

[65] T. M. Sattarri, M. Pal, R. Mirabbasi, and J. Abraham. Ensemble of M5 model tree-based modelling of sodium adsorption ratio. *Journal of AI and Data Mining*, 6(1):68–78, 2018.

[66] D. F. Schwarz, I. R. Konig, and A. Ziegler. On safari to random jungle: A fast implementation of random forests for high-dimensional data. *Bioinformatics*, 26(14):1752–1758, 2010.

[67] T. Shahrukh and P. Kanikar. A survey on decision tree based approaches in data mining. *International Journal of Advanced Research in Computer Science and Software Engineering*, 5(4):613–617, 2015.

[68] P. Taylor and M. Jones. Splitting criteria for regression trees. *Journal of Statistical Computation and Simulation*, 55(4):267–285, 1996.

[69] P. C. Taylor and B. W. Silverman. Block diagrams and splitting criteria for classification trees. *Statistics and Computing*, 3(4):147–161, 1993.

[70] L. Torgo. Functional models for regression tree leaves. In *Proceedings of the 14th International Conference on Machine Learning*, pages 385–393, San Fransisco, CA, 1997. Morgan Kaufmann.

[71] C. Vens and H. Blockeel. A simple regression based heuristic for learning model trees. *Intelligent Data Analysis*, 10(3):215–236, January 2006.

[72] N. Vlahovic. An evaluation framework and a brief survey of decision tree tools. In *Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1299–1304, Opatija, Croatia, 2016. IEEE.

[73] D. S. Vogel, O. Asparouhov, and T. Scheffer. Scalable look-ahead linear regression trees. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 757–764, New York, 2007. ACM.

[74] Y. Wang and I. H. Witten. Inducing model trees for continuous classes. In *Proceedings of the 9th European Conference on Machine Learning*, pages 128–137. Morgan Kaufmann, 1997.

[75] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, USA, 2005.

Appendix A

Acronyms and Abbreviations

This appendix gives a list of acronyms and abbreviations used in this dissertation. The acronyms and abbreviations are listed alphabetically and typeset in bold, with the meaning of the acronym alongside.

AUC	Area under the curve
CART	Classification and Regression Trees
CHAID	Chi-Squared Automatic Detection
CLS	Concept Learning System
DKM	Dietterich, Kearns, Mansour
DM	Distance Measure
GP	Genetic Programming
GPMCC	Genetic Programming for the Mining of Continuous-Valued Classes
HTL	Hybrid Tree Learner
ID3	Iterative Dichotomiser 3
LAD	Least Absolute DeviationS
MPI	Mean Posterior Improvement
MSE	Mean squared error
MTSMOTI	Multi-Target Stepwise Model Tree Induction
OOB	Out-of-bag

OSE	One-sided extreme
OSP	One-sided purity
RETIS	Regression Tree Induction System
RMSE	Root mean square error
sd	Standard deviation
SDR	Standard deviation reduction
SECRET	Scalable Expectation Maximization and Classification based Regression Trees
SMOTI	Stepwise Model Tree Induction
SUPPORT	Smoothed and Unsmoothed Piecewise Polynomial Regression Trees
SVM	Support Vector Machine

Appendix B

Symbols

This appendix lists, alphabetically, the mathematical symbols used in this dissertation, along with their definitions.

a	Predictive variable
A	A set of predictive variables
α	Cost complexity ratio that measures the increase in apparent error rate per pruned leaf
b'	Smoothed predicted value
β	Correction factor for the SDR equation
c	Domain value of a target variable, y
d	Smoothing constant
E	A set of edges in a tree
$\emptyset$	Empty set
f	The number of randomly selected features for splitting a node
F	Total number of features in a random forest
g_R, g_L	Optimal models for the right split and left split respectively
G	Directed graph
i	Index of a value or an attribute in an array or set
j	Dimension index

k	Number of classes in a nominal or dicrete variable
m	Number of observations that do not have missing values
M	Number of trees
n	Number of observations, number of variables
p	Probability
$\phi(P)$	Impurity function
Φ	Impurity
q	Value predicted by regression model at a node
S	Training set
σ	Training set tuples
t	Node
T	Tree
u, e	An ordered pair of nodes
U	A set of nodes
v	Domain value of a nominal attribute
ε	Error rate of a tree
w	Average number of observations that reach a node
x_i	Value of an input attribute at i_{th} position in X
X	Instance space
y	Target attribute
y_i	Value at i_{th} position in y
$\bar{y}$	Sample mean of a target variable
z	Number of model parameters